Etat de l'art sur les techniques de transfert data/audio/vidéo basées Web

Michel Buffa, Alain Giboin, Thierry Bergeron

Etat de l'art

Techniques de transfert data/audio/vidéo basées web

Michel Buffa, *buffa@unice.fr*
Alain Giboin, *alain.giboin@inria.fr*
Thierry Bergeron, *thierry.bergeron@inria.fr*

Laboratoire I3s, Projet AZKAR, Sophia-Antipolis, France

1- Introduction

Le premier objectif, applicatif, du projet AZKAR est d'évaluer 3 types d'usages:

- Des usages de transport impliquant un robot et un télémanipulateur.
- Des usages de santé impliquant un robot, un soignant et le malade.
- Des usages culturels (Musée) impliquant un robot, un conférencier et des visiteurs.

Un second objectif, d'ordre technique, est également, et de manière transversale, d'évaluer la possibilité de commander un robot à distance au travers d'Internet. Comme nous le verrons, les impacts des objectifs techniques (gestion de la bande passante, gestion de l'ouverture des ports, etc) ont beaucoup influé sur les choix d'outils, pour s'orienter au final vers la technologie émergente WebRTC.

Une des contraintes techniques du projet AZKAR est de réaliser des boucles de commande de robots à travers Internet ne dépassant pas 100 ms pour qu'un opérateur à distance ne perçoive pas le retard dû à la transmission. Cette boucle de commande (data) doit être associée à une boucle audio/vidéo à des fins de télé-opération ou de traitement d'images à distance.

Aujourd'hui, la majorité des solutions existantes utilisent des couches de transport et des protocoles spécialisés (comme par exemple les outils vendus par CISCO équipant des salles de conférences privées pour grandes entreprises), mais leur faible inter-opérabilité et la complexité de l'infrastructure matérielle nécessaire est un gros handicap. La plupart utilisent en coulisse des solutions centralisées (la vidéo, l'audio et les datas passent par des "relais") qui ne "scalent pas", ou bien à des prix très élevés.

On trouve également des solutions plus « grand public », comme celles intégrées aux outils de type « messenger » (skype ou google hangouts), basées sur http et utilisant le plus souvent le protocole d'encodage H.264. Aujourd'hui les outils les plus populaires de ce type (Skype, Hangout, FaceTime) sont basés sur des solutions pair à pair (P2P) mais nécessitent l'installation de logiciels spécialisés, parfois non disponibles sur tous les systèmes d'exploitation, et non interopérables.

Ainsi, on peux constater que la transmission de données, audio et vidéo en temps réel sur internet (RTC) n'est ni plus ni moins que de la visioconférence. Un rapide survol des différentes solutions de web-conférence actuelles, grand public comme professionnelles, nous permet de les classer selon 2 points de vue:

- D'un point de vue applicatif:
    - Soit des **applications natives** (ex Skype, Viber, Facetime, Xmeeting),
    - soit des **applications web** (ex Adobe Connect, Google Hangouts),

- D'un point de vue technologique:
    - Soit basé sur des **technologies propriétaires** (Flash, Silverlight...),
    - soit basé sur des **technologies open-source** (WebSocket, WebRTC...).

Ces outils de web-conférence modernes reposent principalement sur le protocole TCP/IP. Ils combinent différents canaux de communication :

- Audio (pont téléphonique, VoIP, etc);
- Vidéo (visioconférence, télé-présence);
- Data (présentation de données de toutes sortes, partage d'applications).

Les solutions de web-conférence comprennent des logiciels et des fonctionnalités qui peuvent différer selon que l'on soit présentateur ou participant. Ainsi certains logiciels peuvent fonctionner comme des applications web (souvent en s'appuyant sur Adobe Flash, Java ou WebRTC), alors que d'autres solutions nécessitent le téléchargement et l'installation du logiciel (ou d'une extension native pour navigateur) sur l'ordinateur ou le terminal de chaque participant et fonctionnent comme une application locale native.

Selon la technologie utilisée, les participants peuvent parler et écouter l'audio sur des lignes téléphoniques standard ou via les microphones et haut-parleurs de l'ordinateur. Certains produits permettent l'utilisation d'une webcam pour afficher les participants tandis que d'autres peuvent exiger un encodage propriétaire ou fourni par une source externe (par exemple à partir d'une caméra vidéo professionnelle connecté via une interface IEEE 1394).

Enfin, ces solutions s'appuient *pour la majorité des cas sur des serveurs intermédiaires* (comme Adobe Connect, très populaire mais payant).

1.1 Le transfert de vidéo en temps réel (ou Streaming)

Un peu d'histoire : la première démonstration de streaming eu lieu en 1968 au Xerox Lab. Présentée par Douglas Engelbart du Stanford Research Institute, cette présentation de 90 minutes fut aussi l'occasion de présenter le premier prototype opérationnel de la souris. C'était la première présentation publique du système NLS (oN-Line System) sur laquelle Engelbart et son équipe travaillaient depuis 1962. (Le système NLS était le premier système d'exploitation multi-utilisateur à proposer liens hypertextes, souris , affichage sur moniteur, fenêtrage, ainsi que d'autres concepts informatiques modernes.)

Depuis les technologies et les protocoles ont beaucoup évolué, mais le principe est resté le même !

Techniquement, le Streaming c'est la lecture d'un flux audio et vidéo au fur a mesure qu'il est diffusé. Cela diffère de la diffusion par téléchargement de fichiers qui nécessite de récupérer l'ensemble des données d'une vidéo avant de pouvoir l'écouter ou le regarder.

Toutefois, la lecture en continu reste, du point de vue théorique, un téléchargement car il y a un échange de données brutes entre un client et un serveur, cependant le stockage est provisoire et n'apparaît pas directement sous forme de fichier sur le disque dur du destinataire. Les données sont téléchargées en continu dans la mémoire vive (RAM), sont analysées (et décompressées) à la volée par l'ordinateur ou le smartphone et rapidement transférées vers un écran ou un lecteur multimédia (pour affichage) puis remplacées par de nouvelles données.

Le flux de données peut être fait en Unicast (vers 1 seul récepteur), en Multicast (un groupe de récepteurs) ou en Broadcast (l'ensemble de tous les récepteurs) pour ce qui concerne le transport des informations (aujourd'hui via les protocoles de bas niveau TCP ou UDP) sur le réseau informatique local ou mondial.

Les transmissions et communications entre serveur et client peuvent utiliser les protocoles suivants de plus haut niveau suivants : RTP, RTSP (standards normalisés par l'IETF) ou MMS (propriétaire Microsoft) ou RTMP (propriétaire Adobe Systems).

1.2 La visioconférence sur les réseaux HTTP

Une autre composante des solutions de web-conférence, le chat texte en temps réel comme par exemple le protocole IRC (Internet Relay Chat) est apparu à la fin des années 1980 (1988 pour IRC). Les premiers web-chats et logiciels de messagerie instantanés sont apparus au milieu des années 90.

En 1996, Microsoft propose le logiciel NetMeeting et introduit la visioconférence en faisant appel au protocole H.323. NetMeeting est l'ancêtre de Windows messenger qui lui même deviendra finalement MSN Messenger.

Cette norme de visioconférence H.323 a été conçue par l'UIT-T (monde des télécoms). Jusque récemment, H.323 était la norme majoritairement utilisée dans le domaine professionnel pour la visioconférence sur IP. Le protocole SIP a pris ensuite le relais, les opérateurs et fournisseurs de matériel ayant intégré l'offre. SIP a été conçu par l'IETF (monde de l'Internet) et sert de support pour la très grande majorité des échanges de voix sur IP.

Ces normes et protocoles décrivent les entités présentes sur le réseau TCP/IP, la signalisation des appels, les codecs audio/vidéo utilisés.

1.3 L'aspect applicatif

Une application native est un "logiciel" que l'on installe sur un terminal (smartphone, tablette, pc) tandis qu'une application web ('webapp') est une application créé spécialement pour les navigateurs, à l'aide des trois langages standards du Web: HTML, CSS et surtout JavaScript.

Les applications natives sont pour la plupart dépendantes du système d'exploitation (une version pour chaque OS) quand elles sont propriétaires. Elles peuvent aussi être dépendantes d'applications et/ou de technologies tierces, comme par exemple les machines virtuelles Java dans le cas de beaucoup de solutions open-source.

Les applications web (qu'on appelle communément "webapps") quand à elles offrent l'avantage de la portabilité et de la sécurité : passer par une communication directe au travers d'un navigateur web permet de s'affranchir en grande partie de la dépendance au système d'exploitation, et de bénéficier d'un environnement de type "sandbox" : une application web ne peut pas lire votre disque dur ou vous espionner ou obtenir des privilèges administrateur sur votre machine, contrairement aux applications natives, surtout lorsqu'elles ne sont pas open source. Ainsi, un robot tournant sous windows xp, vista, 7 ou 8 , Linux ou Unix peut très bien être commandé et communiquer avec un ou plusieurs terminaux, qu'ils soient sous IOS, Androïd, Linux ou Windows.

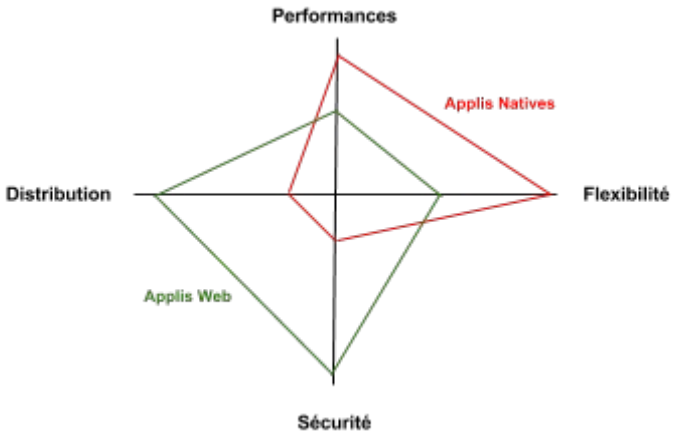
De ce fait, le développement comme la maintenance des clients en est relativement simplifiée, puisqu'en principe il n'est plus nécessaire de développer une version spécifique pour chaque OS ou pour chaque type de client (ordinateur, smartphone, tablette, objet connecté existant ou à venir), mais une version unique et relativement standardisée selon le navigateur utilisé.

Une application native étant propre à un système d'exploitation, le coût de développement et de maintenance est en général plus élevé (bien qu'il existe des systèmes permettant de cross-générer des applications natives). De plus, les processus de soumission sur les magasins en ligne (apple store, Android store, windows store, etc.) étant parfois longs, les mises à jour d'applications peuvent être chronophages.

Une web-application quant à elle fonctionne dans un navigateur internet, et peut donc théoriquement s'exécuter sur n'importe quel environnement disposant d'un navigateur (et aujourd'hui on trouve des navigateurs dans les voitures, les objets connectés, les consoles de jeu, etc.). Sa mise à jour se fait de manière transparente pour l'utilisateur.

Le principal défaut des applications web réside bien souvent dans la performance brute, et dans le fait que les standards du web ne permettent pas encore de réaliser des applications répondant à tout type de besoin. Par exemple, dans le web mobile, les APIs JavaScript permettant d'accéder aux contacts, au bluetooth, aux NFCs, à la carte mémoire, etc, sont en cours de standardisation, mais non encore disponibles sur les systèmes d'exploitation les plus populaires (Android, IOS, Windows Mobile).

Quelle que soit la technologie utilisée, une application native sera presque[1] toujours plus rapide et réactive qu'une web-application. Le schéma ci-dessous compare selon quatre critères les applications natives et les applications web (avec les technologies HTML5).



- Avantages des **applications natives**: meilleures performances, plus de flexibilité dans le développement (accès aux couches basses du système, gestion des priorités des threads, etc). Défauts: sécurité (espionnage, peuvent faire planter le système d'exploitation), distribution (nécessitent une installation, mise à jours plus complexes)

- Avantage des **applications web**: Des mises à jour transparentes, et plus sécurisées, des coût de développement réduit, une distribution et un déploiement aisé multi-plateformes, la continuité dans la navigation de l'utilisateur...

[1] Il existe en effet des compilateurs comme Emscripten de Mozilla, qui compilent des sources C ou C++ directement en JavaScript, en utilisant un jeu d'instructions appelé asm.js, qui produit un code presque aussi rapide que le code C ou C++ original compilé.

1.4 L'aspect technologique des solutions actuelles de Web Conférence

Rappel: Skype, Hangouts, FaceTime etc ne sont pas des solutions basées Web. Nous ne les ignorons pas, mais parlons dans cette section des solutions tournant dans un navigateur Web et ne nécessitant pas d'installation au préalable sur la machine cliente.

Certaines des solutions de web-conférence openSource (comme openMeetings, BigBlueButton) sont basées sur des technologies propriétaires (serveur de média open-source RED5, lui même basé sur Flash…). On trouve principalement parmi ces technologies propriétaires:

La technologie Adobe Flash, propriété d'Adobe, qui se compose d'un environnement de développement intégré (IDE), d'une machine virtuelle utilisée par un player Flash ou par un serveur Flash pour lire les fichiers Flash. Mais le terme « Flash » peut se référer à un lecteur, un environnement ou à un fichier d'application. Ainsi, Flash Player, développé et distribué par Macromedia (racheté en 2005 par Adobe Systems), est une application client fonctionnant sur la plupart des navigateurs web, sous la forme d'un plugin natif devant être installé. Ce logiciel a longtemps été la solution de référence pour la diffusion de flux (*stream*) bi-directionnels audio et vidéo et la transmission de données à faible latence (via les "Flash Sockets", les premiers jeux vidéo d'action multi-participants, c'est grace à ces Flash Sockets !).

La technologie Microsoft Silverlight. Silverlight est un plugin pour navigateur Web multi-plateforme (Windows et Mac OS, Linux via le projet Moonlight), qui permet de développer des applications Web riches dans un moteur de rendu vectoriel. Il fonctionne de façon similaire à Adobe Flash dont il se veut une alternative.

Java avec JavaFX avait essayé également de contrer Flash, mais comme la technologie Silverlight, sans arriver à s'imposer.

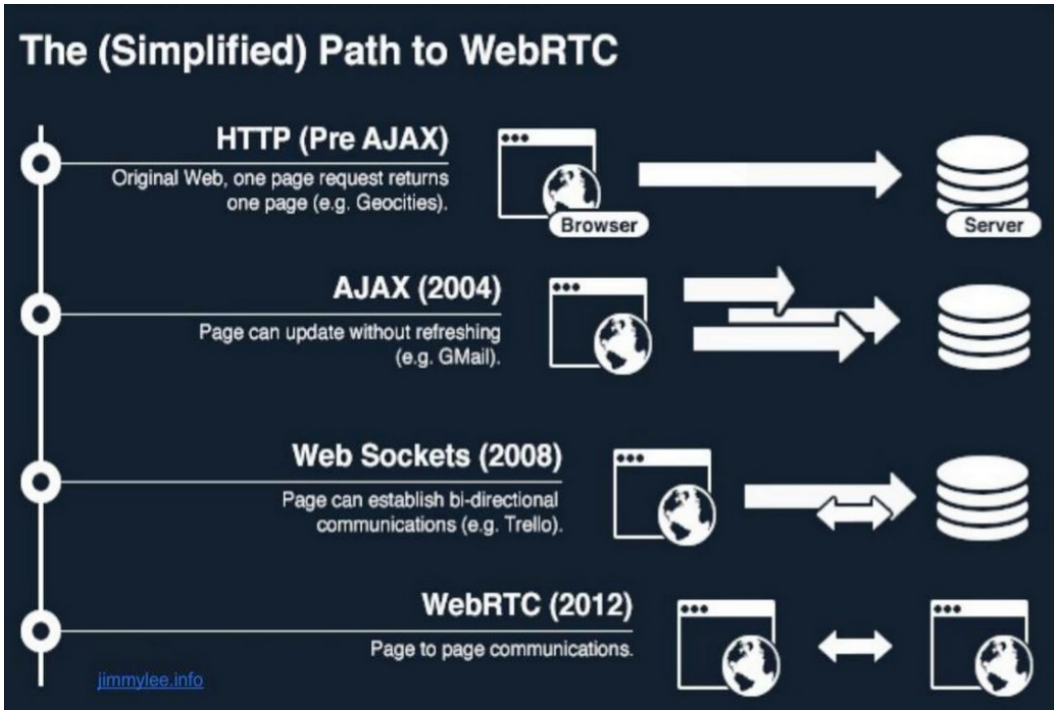
A savoir que tous ces plugins sont des mauvais citoyens du web, l'interaction avec les standards du web comme CSS est quasiment inexistante, il n'est pas possible de sélectionner un texte à la souris dans une application Flash, Silverlight ou JavaFX. Ces plugins sont juste "un rectangle propriétaire dans une page web". A noter que Flash et ces technologies ne sont pas supportées ni par IOS ni par Android aujourd'hui.

Parmi les technologies open-source :

Web Sockets: Le W3C a introduit à côté d'HTML5 (et à peu près en même temps) l'API des Web Socket, alternative aux Flash Sockets d'Adobe. C'est un canal de communication full-duplex, en mode connecté (contrairement à Ajax) entre navigateurs serveurs de Web Sockets. Aujourd'hui la plupart des serveurs HTTP les plus populaires intègrent le moyen d'en faire également des serveurs de Web Sockets. Le protocole des Web Sockets est beaucoup moins verbeux que HTTP et autorise des temps de latence

bien moins élevés qu'avec la technologie Ajax (y compris Ajax long polling/COMET qui est une version de ce protocole permettant de conserver des communications ouvertes plus longtemps). On trouve de nombreux jeux d'action multi-participants sur le web aujourd'hui, basés sur des communications par Web Sockets. Les Web Sockets nécessitent de passer par un serveur centralisé, ce qui rend la gestion de la charge compliquée dès que la quantité de personnes connectées et de messages échangés devient importante.

WebRTC: Web Real-Time Communication (http://www.webrtc.org/) est un standard du W3C. Ce standard se compose de protocoles et d'APIs JavaScript permettant la communication pair-à-pair audio, vidéo et data entre navigateurs web et/ou applications natives.



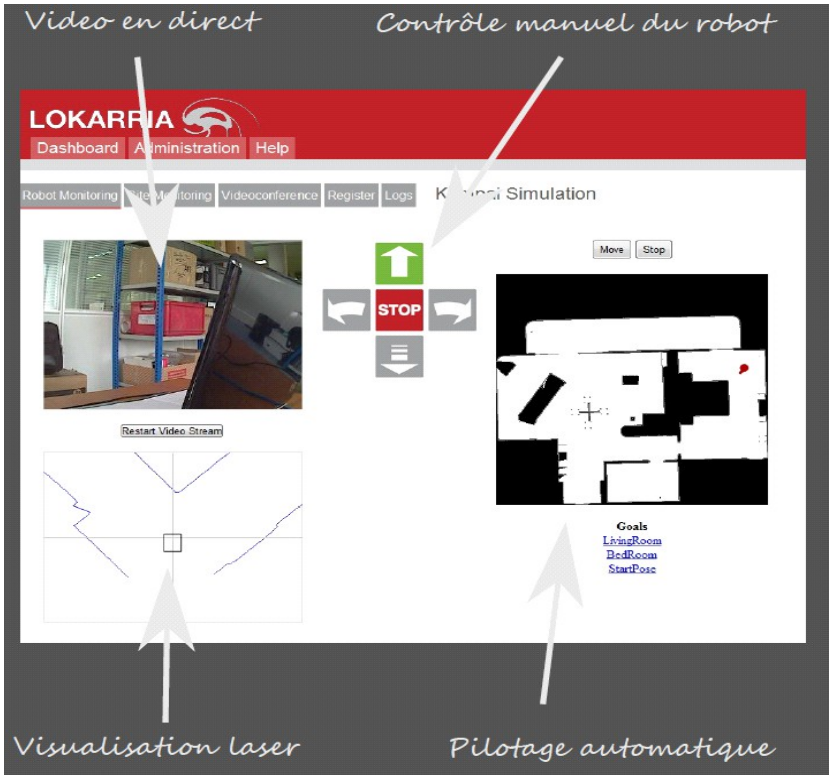
Historique des standard du web permettant des communication temps réel, d'AjaX jusqu'à WebRTC.

1.5 Technologies de visio/contrôle robot en usage chez Robosoft

Robosoft a choisi de développer ses robots en C# s'appuyant sur les technologies mises à disposition par Microsoft, à savoir le framework .Net et plus précisément le service de robotique Microsoft Robotics Developer Studio (MRDS) qui facilite l'implémentation de logiciel de robotique temps réel. Afin d'ouvrir les communications avec les robots autrement que par ces services C#, Robosoft a mis en place un protocole de communication HTTP avec ses robots : LOKARRIA.

Lokkaria regroupe un ensemble de services MRDS ouvert au protocole HTTP. Il permet donc une communication avec n'importe quel type de logiciel capable de générer des requêtes HTTP. Plus précisément, le format d'échange de données se fait en JSON que ce soit pour récupérer des informations du robot ou pour le contrôler.

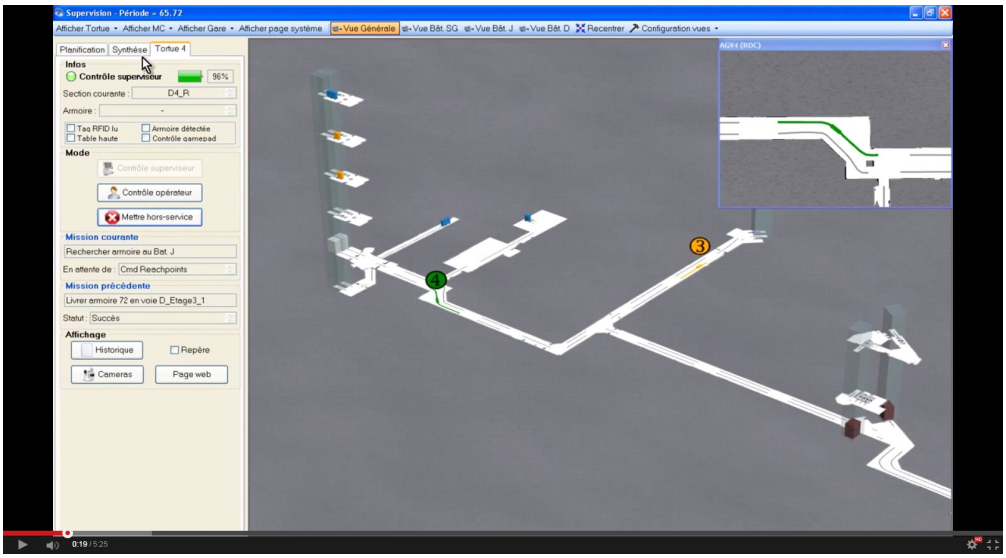
En effet, l'envoi de requêtes GET permet d'accéder à l'état du robot : sa vitesse, la carte générée et sa position sur celle-ci, le retour du laser d'anticollision et des cameras… Tandis que les requêtes POST permettent de donner des ordres de déplacement au robot. Il faut distinguer deux types de contrôle du robot. Il y a le contrôle manuel où on peut faire avancer, reculer ou tourner le robot, et le contrôle automatique où l'on désigne un point de destination sur la carte du robot et celui-ci génère sa propre trajectoire pour atteindre son but. La première application du service Lokkaria a été une application web de contrôle du robot dont voici une capture d'écran :



On peut y voir le retour de la camera, du télémètre laser et de la carte du robot ainsi que des boutons de contrôle du robot.

Le Superviseur de Flotte

Afin de pouvoir gérer simultanément plusieurs robots, Robosoft a développé un logiciel de supervision de flotte qui permet un monitoring de l'état de différents robots sur une seule interface utilisateur. Ce logiciel a pour but la complète autonomie du système, mais elle reste sous contrôle éventuel des opérateurs. Pour cela, l'opérateur possède une interface qui lui indique l'état des robots, leurs positions sur la carte, le planning de leur mission…

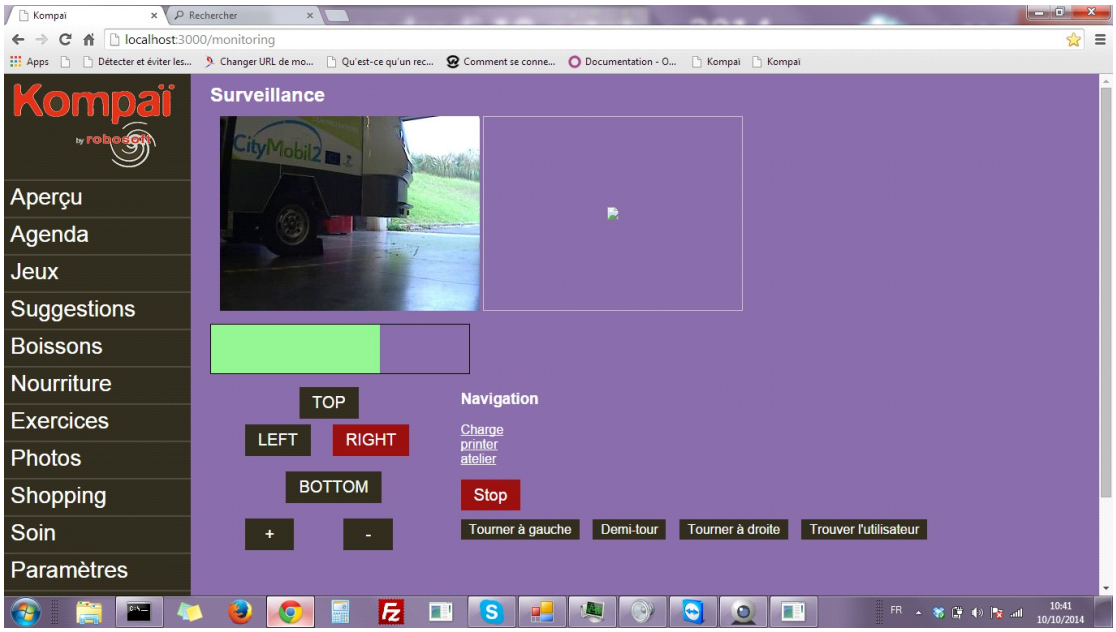


Sur ce même logiciel, on retrouve une interface de contrôle du robot avec les retours de la camera et les boutons pour controler le robot manuellement (le contrôle en mode automatique est toujours possible)

Les échanges d'informations entre le logiciel de supervision et les différents robots se font via un réseau virtuel privé (VPN) et s'appuie sur les services Lokkaria pour fonctionner.

Kompaî

Kompai est un robot d'assistance aux personnes agées, qui peut faire de la téléprésence. C'est-à-dire qu'un opérateur peut prendre le contrôle du robot à distance, et déclencher une Visio conférence pour faire de la levée de doute, mais également déplacer le robot dans son environnement pour rechercher la personne si celle-ci ne décroche pas. Pour cela, il a à sa disposition une interface web qui est situé à l'intérieur du robot. En effet, via cette interface il peut éditer l'agenda de la personne âgée et le contenu de l'application situé sur le robot tel que ses rappels de médicaments, sa liste de courses…



Cette interface web inclus également une interface de contrôle à distance du robot similaire à celle faite pour Lokarria, et à le retour visuel des deux cameras ip afin de pouvoir détecter si la personne est tombé ou non. Le robot ayant des points préenregistrer sur la carte, l'opérateur peut y envoyer le robot d'un simple clic pour faciliter les déplacements.

Puisque le contenu du robot est localisé sur celui-ci, l'opérateur peut utiliser le logiciel de prise de contrôle d'ordinateur « teamviewer » pour accéder à la tablette situé sur le robot et le piloter à distance.

1.6 Technologie pour la visite de musée en usage chez Another World

Actuellement, une visite d'un musée par web conférence se compose de plusieurs supports :

- Un partage audio vidéo entre le conférencier et la ou les classes
- Un partage d'image
- Un partage de fichier sonore
- Un partage de vidéo préenregistrée
- Un partage de flux vidéo venant des caméras du musée pilotées par le conférencier

Tout cela au travers de la plateforme collaborative adobe Connect, qui est, pour rappel, une solution propriétaire et centralisée permettant le dialogue audio vidéo, le partage d'images, de fichiers sonores et vidéos et le partage d'écran.

Pour la vidéo des caméras pilotables, une première solution est de partager l'écran de ce que voit le manipulateur dans son application de contrôle, mais on obtient une perte de fluidité et de définition d'image.

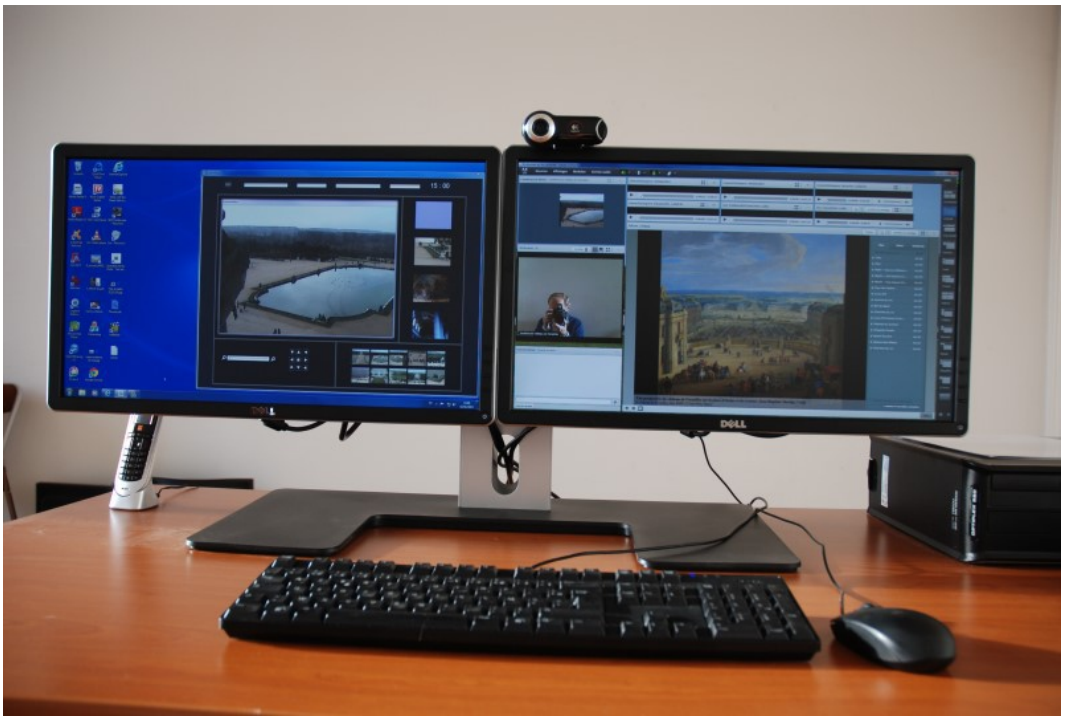
Une seconde solution est de streamer le flux d'une des caméras par sélection du conférencier vers un CDN[2] et d'intégrer sur la plateforme un lecteur vidéo flash. On gagne ainsi en fluidité et en définition. Toutefois, il n'y a pas de flux multiple envoyé vers chaque participant. Seul le PC d'encodage vers le CDN voit tous les flux de toutes les caméras.

Dans ces visites l'accent est mis sur l'interactivité entre le conférencier et la classe. Il faut donc un outil qui permet l'échange audio vidéo de manière simple. Ensuite la diversité des supports permet de rendre la visite vivante avec, pour point d'orgue, l'utilisation des caméras "live" qui immerge la classe dans le musée avec toujours des imprévu, comme le temps, des travaux, des visiteurs dans le champ, …

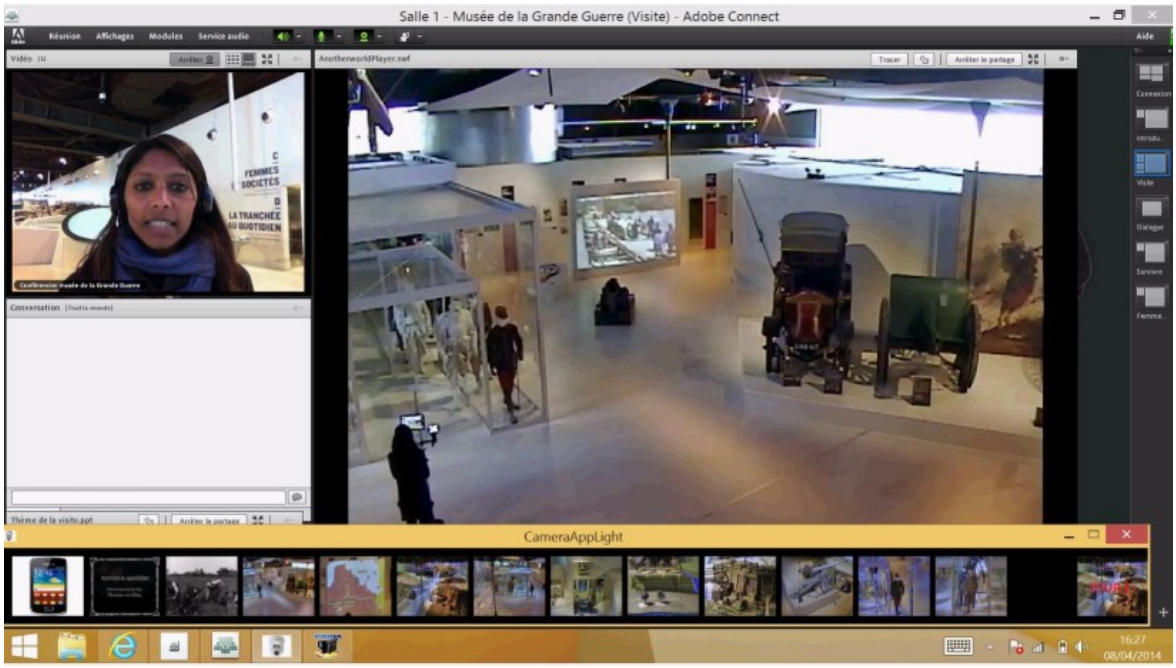
Une plate-forme de type web RTC permettrai aisément de rendre ces mêmes services de communication audio vidéo et même d'intégrer un tchat. Pour le partage de document de type image, vidéo et sonores, une simple synchronisation des pages web du conférencier et de la classe par un canal data pourrait suffire. Pour ce qui est du flux vidéo des caméras la solution de streaming telle qu'évoquée plus haut reste la meilleure.

Toutefois, dans le cadre de visites non plus d'un groupe de visiteurs (classe, maison de retraite, ...) mais d'internautes individuels, un problème dit de "full mesh" apparaîtrait très vite, à savoir une saturation de la bande passante et une explosion du nombre de ports ouverts sur le poste du conférencier. S'il y à beaucoup de connexions audio vidéo, l'utilisation d'un MCU[3] comme nous le verrons plus loin devrait régler ce genre de soucis.

[2] CDN: Content Délivery Network
[3] MCU: multipoint control unit



Installation double écran au château de Versailles. L'écran partagé adobe connect est à droite, l'application de pilotage des caméras sur l'écran de gauche.



Copie d'écran de la tablette surface utilisée au musée de la Grande Guerre de Meaux. La salle Adobe Connect intègre un player vidéo, dans lequel le flux live streamé est lu.

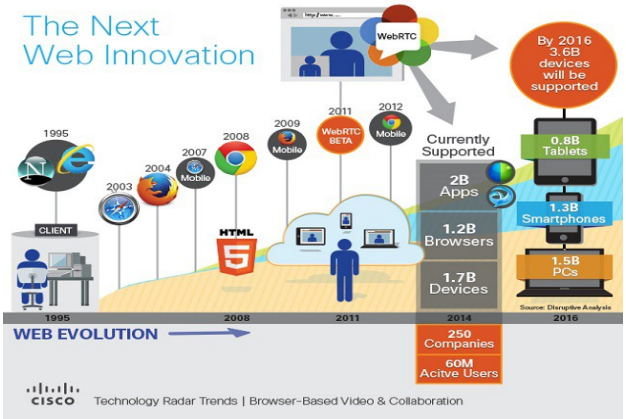
2 - Un nouveau standard pour la video/audio/data P2P : WebRTC

WebRTC est le tout premier standard élaboré au W3C et à l'IETF pour normaliser les communications en temps réel peer-to-peer. Concrètement, il couvre les appels *en streaming* vidéos, audio, et les échanges de données. Destiné au départ aux navigateurs, les développeurs se sont déjà emparés du code open source et des spécifications pour réaliser des implémentations des APIs dans d'autres langages comme C#, Java, ou Swift, permettant le développement natif d'applications, y compris sur iOS, Android ou Windows Mobile. Notons qu'en Janvier 2015 il n'est pas possible de faire du WebRTC dans un navigateur web sur Windows Mobile ou sur IOS.

.

Skype (Microsoft) ou FaceTime (Apple) sont d'excellents produits propriétaires, mais dont le fonctionnement reste totalement obscur. À l'inverse, WebRTC est une technologie qui permet de réaliser les principales fonctionnalités de ces produits, au cœur même d'autres applications ou sites internet. Google est à l'origine de WebRTC en ayant ouvert le source de la technologie derrière la partie audio et vidéo de Google Talk (devenu par la suite Google Hangouts).

Depuis juin 2014, Hangouts fonctionne nativement dans Chrome grâce à WebRTC, mais les autres navigateurs ont toujours besoin d'un plug-in. Mozilla a récemment lancé Firefox Hello, un service de communication basé sur WebRTC. Toujours en 2014, Cisco indiquait[4] que parmi les principales tendances, l'utilisation du navigateur Web comme client de communications unifiées et de collaboration (grâce à HTML5 / WebRTC) allait bouleverser l'usage des outils collaboratifs en entreprise ou dans les relations BtoB et BtoC. Enfin, bien plus que l'évolution des usages, c'est aussi le nombre de périphériques "capables de" qui va évoluer rapidement, avec en 2016, une projection de 3,6 milliards de terminaux communiquant nativement sans installation de logiciel préalable.



Timeline évoquant l'évolution des terminaux web vers WebRTC. (Source: CISCO)

[4] Source: http://gblogs.cisco.com/fr-collaboration/2014/06/cisco-devoile-les-principales-tendances-it-pour-les-entreprises-web-rtc-html-5-au-coeur-des-enjeux/

2.1 Les origines de WebRTC

L'idée de WebRTC est née fin 2009, un an après le lancement de Google Chrome. En répertoriant les fonctionnalités manquantes dans son offre Web, Google constate que la plupart d'entre elles font déjà l'objet de projets, mais qu'il il n'y à pas de solutions pour les communications en temps réel (RTC).

A cette époque, seul Flash ou ActiveX de Microsoft (via des plugins propriétaires) permettent de faire du RTC. Flash offrant une faible qualité sur mobile et imposant une licence côté serveur. Quant aux autres plugins, ils se révèlent délicats à installer et leur fonctionnement sur différents navigateurs et sur plusieurs systèmes d'exploitation pose problème.

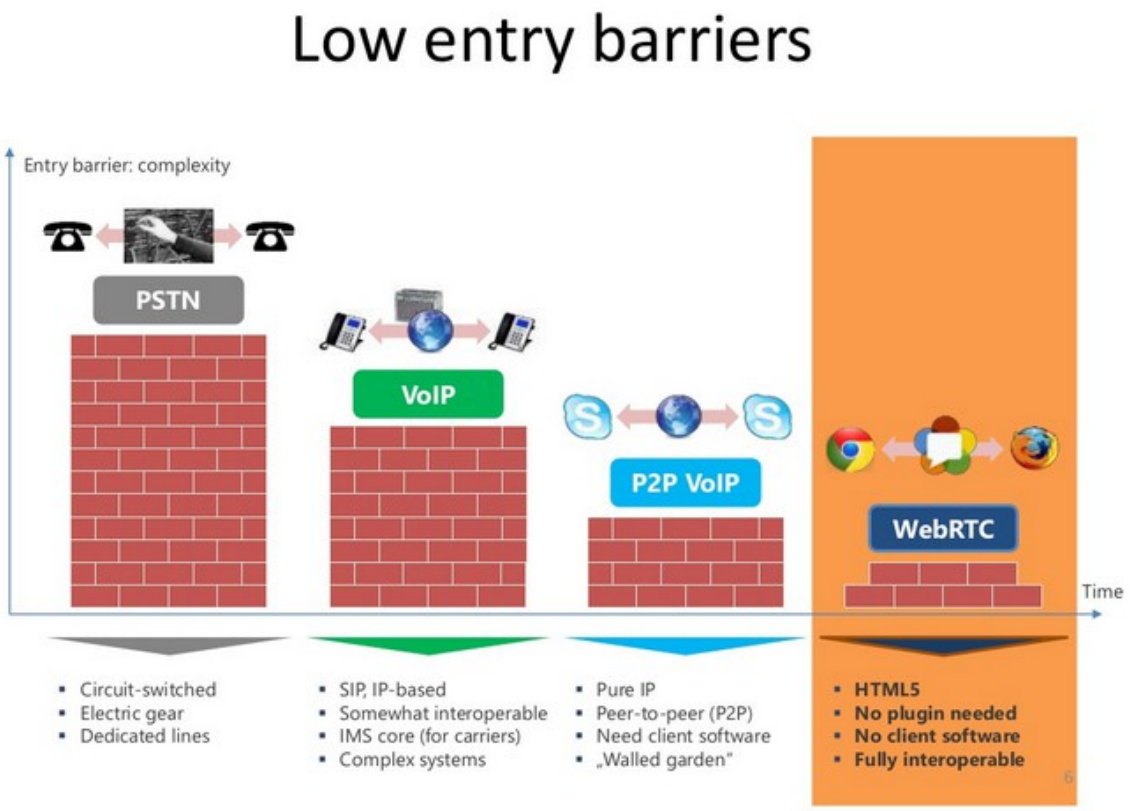
Début 2010, Google rachète la firme On2 Technologies, spécialisée dans la compression de vidéo de haute qualité. On2 à notamment développé les architectures des codecs VP3, VP4, VP5, TrueMotion VP6, TrueMotion VP7 et VP8. On2 positionnait ses codecs comme une alternative gratuite face aux licences payantes des codecs de la série H.2xx (dont le fameux H.264) qui sont largement utilisés, brevetés et standardisés.

Google libère alors les sources du codec vidéo VP8 sous le nom de WebM, L'objectif étant de remplacer le codec H.264 pour les vidéos web et de diminuer les coûts de licence pour tous, mais aussi et surtout pour Google lui-même.

Courant 2010, Google rachète GIPS, connu pour ses frameworks facilitant le développement d'applications VOIP et de visioconférence. GIPS produit aussi des composants audio/vidéo bas-niveau utilisés par non seulement par Google, mais aussi par Skype, AOL, Yahoo, Cisco et bien d'autres.

En 2011, à l'initiative de Google qui ouvre les sources des technologies de GIPS, des groupes de travail au W3C pour la standardisation d'une solution de RTC sur le Web sont créés. Rapidement, d'autres grands acteurs comme Mozilla, Voxeo, Ericsson, Intel, Cisco, Samsung, France Telecom, AT&T, Avaya, Huawei et Opera, participent à la naissance du standard WebRTC. Depuis, et par itérations successives, l'écriture du standard avance. Il est publié sur le site du W3C et du code de référence est disponible en open source.

En augmentant "les possibilités des applications web", ces entreprises "captent" plus de clients potentiels via leurs services annexes, et poussent un peu plus le curseur vers le "tout web" qui est leur fond de commerce.



Historique des solutions de voix et vidéo sur IP

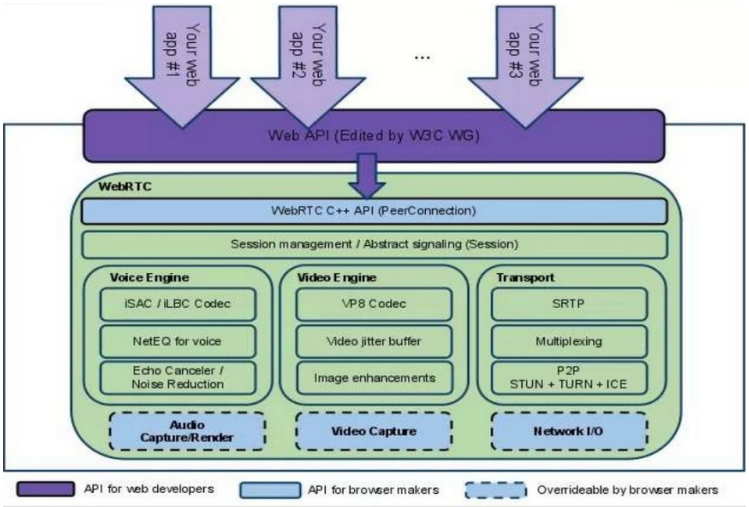
2.2 Qu'est-ce que WebRTC

WebRTC (où RTC signifie Real-Time-Communication) est une collection d'interfaces de programmation (API) JavaScript dont les spécifications font l'objet d'un standard du W3C (actuellement à l'état de Working Draft, donc encore sujet à changements, voir http://www.w3.org/TR/webrtc/). Ce standard se compose de documents et d'implémentations de référence à destination des fabriquant de navigateurs. Nous ne nous intéresserons pas à cet aspect très particulier dans ce document, l'écriture d'un navigateur compatible WebRTC n'étant pas un des thèmes du projet AZKAR. En revanche, le développement d'applications utilisant WebRTC est au coeur du projet.

Pour le développeur d'applications, le standard comprend trois APIs JavaScript :

- **MediaStream** (l'API de flux réseau) également appelée getUserMedia(), qui gère le flux de données audio ou vidéo en provenance de la caméra ou du micro de l'utilisateur. A noter que cette API prévoit le streaming de données capteurs, de fichiers, l'enregistrement des flux sur disque, mais que ces parties n'ont pas encore été spécifiées. Quelques exemples de ce que permet de faire aujourd'hui cette API:
    - Ajouts d'effets à une webcam: http://webcamtoy.com/fr/app/
    - Jeu de suivi de visages: http://shinydemos.com/facekat/
    - Génération d'image ASCII: http://idevelop.ro/ascii-camera/

- **RTCPeerConnection** (l'API de Connexion), qui permet à plusieurs utilisateurs de communiquer via leurs navigateurs. C'est l'API qui permet une communication stable et efficace de flux *streamés* audio/vidéo entre pairs. Pour le développeur JavaScript, RTCPeerConnection cache toute la plomberie : réduction de bruit et d'écho, adaptation à la bande passante, codecs etc.



Les APIs WebRTC (en violet) cachent toute la plomberie de bas niveau (en vert).

- l'API **RTCDataChannel** qui permet la communication d'autres types de données que l'audio et la vidéo, avec une faible latence et de manière sécurisée. Cette API permet de configurer le mode "reliable" ou "unreliable" et, comme les Web Sockets, peut contenir n'importe quel type de données (objets JSON), par exemple pour le jeu en temps réel, le dialogue en ligne, le transfert de fichiers, le contrôle de machine à distance, etc.

  A noter une limitation: comme pour les Web Sockets il y a une taille maximale de "trames" que l'on peut envoyer: 64 kilo octets. Pour envoyer un fichier ou une image par ce biais il faut découper le fichier source en "chunks" et les recoller à l'arrivée. On trouve des librairies et de nombreux exemples pour effectuer cette opération.

2.3 Principes de fonctionnement

Une solution WebRTC se compose de deux parties:

1. Une partie cliente: qui est le plus souvent une application web tournant dans un navigateur, codée en JavaScript/HTML/CSS. Il existe également des SDK pour développer des clients "natifs", en particulier pour apporter des solutions WebRTC là où les navigateurs web ne supportent pas encore cette technologie (IOS/Safari en particulier).

2. Une partie "serveur": on trouve ici deux éléments distincts qui peuvent ne pas tourner sur la même machine: 1) un serveur dit de "signaling" qui sert à échanger entre des clients leurs configuration (dont leurs IPs externes, notamment), et 2) un serveur de "relais" permettant d'effectuer les échanges vidéo/audio/data lorsqu'une connexion P2P ne peut être réalisée directement entre deux clients. Les termes utilisés dans le jargon WebRTC sont "serveur STUN" pour le signaling, et "serveur TURN" pour le serveur de relais.

Prenons le cas le plus classique d'une application WebRTC cliente tournant dans un navigateur web. On parle ici de "webapp WebRTC cliente".

Cette application web est écrite en HTML/CSS/JavaScript. La partie HTML/CSS sera utilisée pour les IHMs et la partie JavaScript pour la communication avec les serveurs (signalisation/STUN et relais/TURN) et pour l'appel de fonctions WebRTC (établissement de la connexion, envoie des données, etc)

Coté client, les étapes pour établir une communication sont:

1. Récupérer des flux audio et/ou vidéo, choisir la source (caméra av/arrière, micro, etc.),

2. Obtenir son adresse IP et son port "externes". Si le client est dans un réseau d'entreprise ou dans une maison, il a une adresse externe qui est celle de son routeur/box et son adresse interne est différente. Il existe des règle de NAT (association adresse externe+port/adresse interne+port) dans les routeurs.

3. Echanger ces IP avec les autres clients WebRTC (pairs) pour permettre la connexion, même à travers les pare-feu et NATs[5]. Les clients communiquent via leurs adresses externes, ils doivent donc les exposer. Le serveur de signaling sert à effectuer cette tâche d'échange.

4. Coordonner la prise de contact entre pairs et signaler les erreurs, démarrer ou fermer des sessions.

5. Échanger des informations sur les capacités du client, telles que résolution et codecs.

6. Streamer en p2p audio et/ou vidéo, échanger des données...

Coté serveur, l'objectif est de mettre en communication les pairs et de transmettre les messages de signaling. Le serveur de signalisation est obligatoire, mais WebRTC n'en impose aucun de particulier. Les technologies de traversée de NAT utilisées pour les connexions de pairs à distance sont ICE[6], STUN[7], TURN[8]..

[5] En réseau informatique, on dit qu'un routeur fait du ***Network Address Translation*** (NAT) (« traduction d'adresse réseau ») lorsqu'il fait correspondre les adresses IP internes non-uniques et souvent non routables d'un intranet à un ensemble d'adresses externes uniques et routables. Ce mécanisme permet notamment de faire correspondre une seule adresse externe publique visible sur Internet à toutes les adresses d'un réseau privé, et pallie ainsi l'épuisement des adresses IPv4

[6] ICE (Interactive Connectivity Establishment): framework complet de prise en charge des connexions

[7] STUN (Session Traversal Utilities for NAT): serveur qui effectue la mise en relation

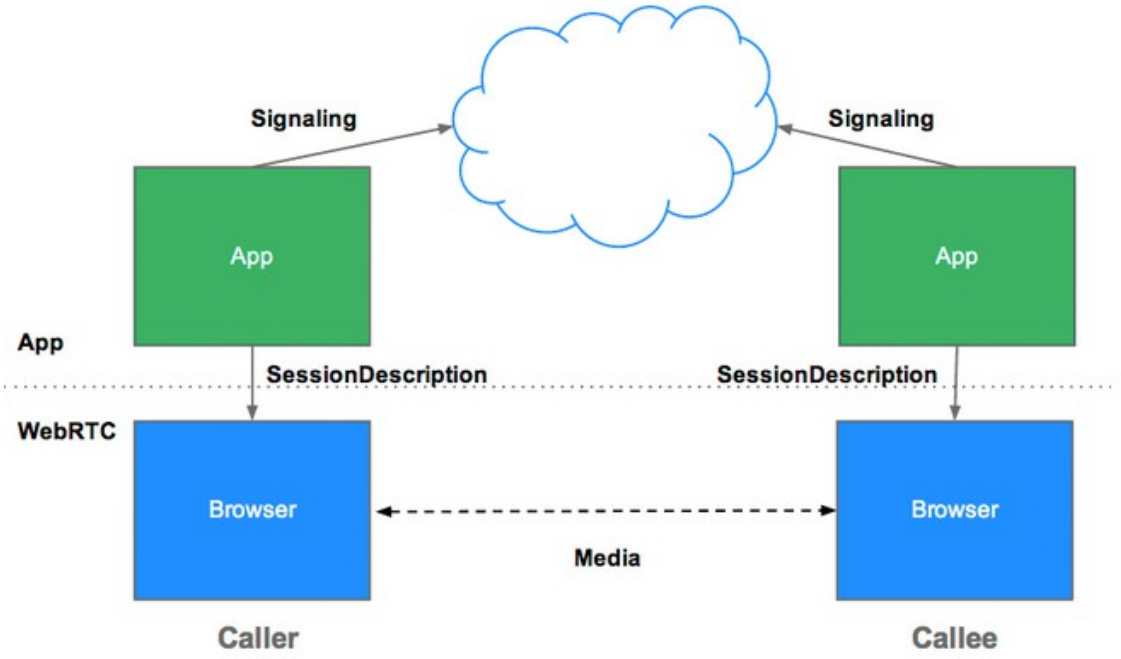
[8] TURN (Traversal Using Relays around NAT): intermédiaire entre les pairs pour le passage de NATs ou de firewalls

2.3.1 Le signaling

Le *signaling* est le mécanisme qui coordonne la communication et permet d'envoyer et recevoir des messages de contrôle. Si WebRTC ne définit pas les méthodes et protocoles de signalisation et impose de passer par un serveur tiers, c'est pour de très bonnes raisons:

1. Pour éviter la redondance et maximiser la compatibilité avec les technologies établies. En effet, différentes applications peuvent nécessiter différents protocoles et le groupe de travail WebRTC ne voulait pas se verrouiller sur un protocole qui pourrait se révéler insuffisant à couvrir toutes les utilisations possibles.

2. WebRTC s'exécute dans un navigateur et le signaling implique que celui-ci soit "*stateful*", c'est à dire qu'il enregistre les états (l'inverse de "*stateless*"). Or cela devient problématique si la signalisation est perdue a chaque foi que la page est rechargée…



Architecture JSEP

L'architecture de JSEP[9] évite au navigateur d'avoir à enregistrer l'état, c'est à dire de fonctionner comme une machine à état de signalisation. Ainsi, plus de perte de données de signalisation à chaque rechargement de la page. Au lieu de cela, l'état de signalisation est enregistré sur un serveur tiers.

[9] JSEP (JavaScript Session Establishment Protocol):

A noter que WebRTC impose quand même l'usage du format SDP[10] au serveur de signaling. Même si des discussions sont en cours pour le remplacer par du JSON[11], à ce jour, SDP reste le seul et unique choix possible.

Le signaling sert à échanger trois types d'information: Les messages de contrôle de session (initialiser les communications et rapporter les erreurs), les configuration réseau pour le monde "extérieur", (adresse IP, ports..), les capacités "médias" (résolutions et codecs supportés par les devices / browsers ,etc…). Sans rentrer dans les détails, on peux résumer le processus de signalisation en 3 phases:

1. Connexion au serveur de signalisation:

   La méthode la plus courante consiste pour les utilisateurs à visiter le même site Web et se connecter via un serveur de signalisation partagé. Les clients échangent une sorte de clef ou un jeton qui permet l'identification unique de la session. Ce jeton partagé peut être un numéro ou un nom de room, une URL ou un ID de conversation.

   Une solution assez classique utilisée pour les clients WebRTC pour se connecter à un serveur de signalisation est de passer par l'API Web Socket. Le signaling ne provoque pas de charge énorme sur le serveur, la taille des informations à échanger étant réduite. Cependant pour des applications pouvant impliquer des milliers de mises en relation simultanée, des solutions comme l'API de message de Google App Engine (solution cloud de Google), ou les serveurs hébergés sur le coud Amazon sont souvent proposée par les offres WebRTC commerciales.

2. Échange des informations réseau:

   Cette étape est apellée "finding candidates". Son but est de permettre aux navigateurs web d'échanger les informations réseau nécessaires pour échanger des flux médias en P2P. Comme la plupart des clients utilisent des adresses IP privées, il est nécéssaire de faire une translation d'adresse par NAT. Pour trouver une adresse IP adressable, WebRTC utilisera des serveurs STUN et/ou TURN. Ces serveurs fournissent un client avec une adresse IP qui peut être partagée avec ses pairs pour les connexions de médias.

   WebRTC appelle ce processus d'utilisation de serveurs STUN et TURN l'Interactive Connectivity Establishment (ICE). ICE tentera d'abord d'utiliser STUN et si STUN n'est pas possible, TURN sera utilisé.

[10] SDP (Session Description Protocol): description des métadonnées de la connexion (résolution, format, codecs.)
[11] JSON (Javascript Object Notation): Format de données textuelles dérivé de la notation des objets du langage javascript.

3. Négociation des sessions média:

   Une fois que les clients savent comment communiquer entre eux, ils doivent s'entendre sur le type et le format des médias. C'est ici qu'entre en jeux l'architecture JSEP et le protocole SDP pour identifier le codec, la résolution, le bitrate, la taille, etc. des supports pris en charge par le client.

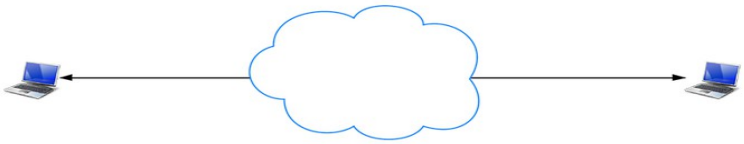
Le processus de signalisation terminé avec succès, le Streaming proprement dit peut commencer: Les données sont alors transmises directement en peer to peer entre l'appelant et l'appelé, ou, si cela échoue, via un serveur relais intermédiaire (au moyen d'un serveur TURN).

2.3.2 Le framework ICE et les protocoles STUN et TURN

Les applications WebRTC utilisent le framework ICE pour surmonter les difficultés de la mise en réseau du monde réel: la plupart des dispositifs en mesure d'utiliser WebRTC vivent derrière une ou plusieurs couches de NAT et peuvent avoir des couches de sécurité qui bloquent certains ports et protocoles (parfois avec de l'inspection profonde de paquets IPs ou "Deep Paquet Inspection DPI"). De plus, beaucoup d'entre eux sont derrière des proxies et des firewalls d'entreprise, sans compter les firewalls et NAT des routeurs Wifi domestiques.

Le protocole STUN et son extension TURN sont utilisés par ICE pour traverser les NATs et se conformer aux aléas du réseau (DPI[12], etc.). ICE tente de connecter directement les pairs. Il essaie toutes les possibilités en parallèle et choisit l'option la plus efficace qui fonctionne.

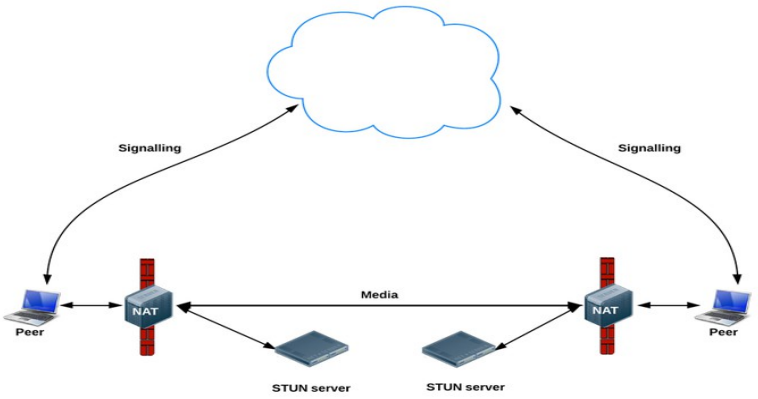
ICE tente d'abord d'établir une connexion directe entre les pairs, avec la plus faible latence possible, en utilisant l'adresse hôte obtenue à partir du système d'exploitation du pair et de sa carte réseau.



Connexion directe entre pairs, sans NAT ni Firewall

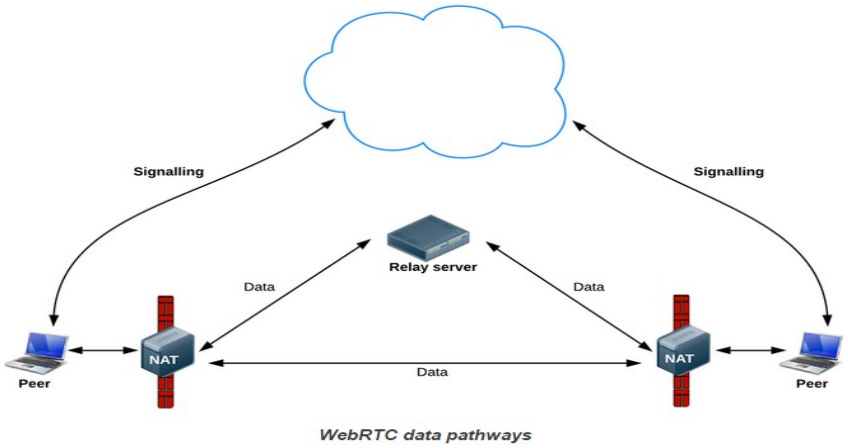
En cas d'échec, (NAT ou Firewall…) ICE passera par un serveur STUN pour obtenir une adresse réseau externe. Dans ce processus, les serveurs STUN ont une seule tâche: permettre à un pair derrière un NAT de découvrir son adresse publique et son port. (Google fournit des serveurs STUN publics). Si UDP échoue, ICE tentera TCP: avec HTTP, puis HTTPS. Selon webrtcstats.com, 85% des appels WebRTC établissent avec succès leurs connexions via des serveurs STUN.

[12] DPI (Deep Paquet Inspection): Analyse du contenu (au-delà de l'en-tête) d'un paquet réseau (paquet IP le plus souvent) de façon à en tirer des statistiques, à filtrer ceux-ci ou à détecter des intrusions, du spam ou tout autre contenu prédéfini. Le DPI peut servir notamment à la censure sur Internet ou dans le cadre de dispositifs de protection de la propriété intellectuelle.



Utilisation de serveurs STUN pour obtenir les IP publiques (ip:port)

Compte-tenu de la simplicité de leur tâche et du peu de mémoire dont ils ont besoin, les serveurs STUN consomment peu de bande passante et peuvent gérer un grand nombre de demandes et supporter de fortes charges.



WebRTC data pathways

Si la connexion directe échoue (firewall, DPI, etc...) le trafic est alors routé via un serveur relais TURN. Il existe des serveurs publics (Google...). Les serveurs TURN ont un rôle simple: Relayer un flux, mais contrairement aux serveurs STUN, ils consomment de la bande passante.

Pour rappel, TURN n'est utilisé que pour relayer le flux Data/Audio et Video en streaming, pas les données de signalisation. Cet aspect est l'un des **points critiques à prendre en compte** dans le cadre du projet AZKAR: Si l'on envisage de déployer un nombre important de robots, le cas où la partie TURN doit relayer quelques dizaines ou centaines de communications vidéos et audio risque de poser un problème de charge. C'est ici que sans doute des gateways (passerelles) ou des solutions commerciales (que nous verrons plus loin) seront nécessaires.

2.3.3 Le Multicast et la gestion de bande passante

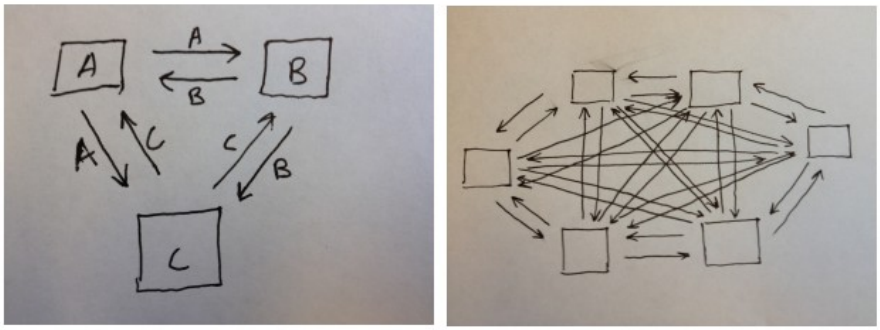
Tel qu'elle est actuellement implémentée, l'api RTCPeerConnections de WebRTC ne prend en charge que les communications One-To-One, or, comme déjà évoqué, le projet AZKAR comporte 3 évaluations:

- Transport (Mode Unicast - One-To-One): Le robot, le télémanipulateur.
- Santé (Mode Multicast - One-To-Many): Le robot, le soignant, le malade.
- Musée (Mode BroadCast - One-To-Many): Le robot, le conférencier, le ou les visiteurs.

Dans le cadre d'un usage One-To-Many (ou Multicast), la gestion de la bande passante vas rapidement poser un problème délicat: Prenons le cas où 1 Robot est manipulé par 1 conférencier qui interagit avec 10 visiteurs individuels, chacun dans un lieu séparé des autres, cela implique d'ouvrir jusqu'à 42 ports simultanément sur le navigateur du conférencier:

- 1 port RTP entrant (flux audio en provenance du robot)
- 1 port RTP entrant (flux vidéo en provenance du robot)
- 10 ports RTP sortants (flux vidéos vers chaque visiteur)
- 10 ports RTP sortants (flux audio vers chaque visiteur)
- 10 ports RTP entrants (flux vidéos en provenance de chaque visiteur)
- 10 ports RTP entrants (flux audio en provenance de chaque visiteur)

Une application WebRTC peut utiliser plusieurs instances de l'API RTCPeerConnections. Ainsi chaque noeud final se connecte à chaque autre point d'extrémité dans une configuration de réseau. C'est l'approche adoptée par des applications telles que talky.io (https://talky.io) et qui fonctionne remarquablement bien pour un petit nombre de pairs. Au-delà, la charge CPU et la consommation de bande passante devient excessive, en particulier pour les clients mobiles.

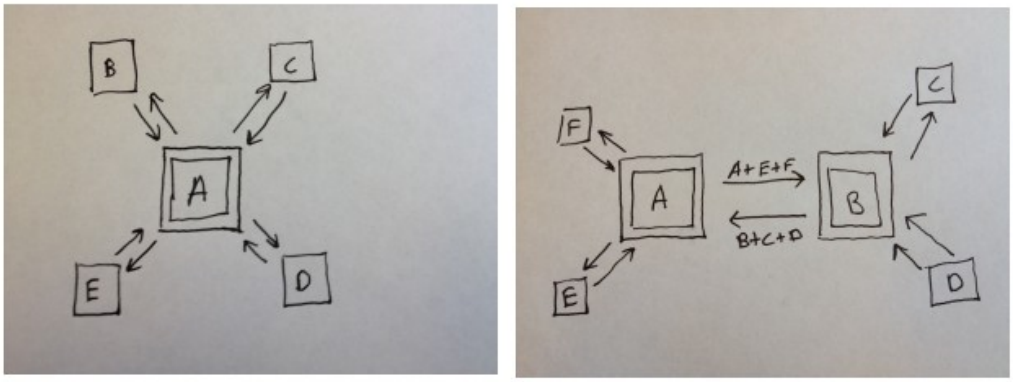


Topologies "Full Mesh": Tout le monde est connecté à tout le monde

Depuis Chrome 31 et Opera 18, l'objet "Stream" (flux) d'un RTCPeerConnection peut être utilisé comme entrée pour un autre. Cela autorise des architectures plus flexibles, car il permet à une

application Web de gérer le routage des appels en choisissant les autres pairs auquel se connecter.

Ainsi, avec une architecture en étoile, l'application WebRTC choisit un point terminal qui redistribue les flux audio et vidéos à tous les autres clients. Le point terminal A étant toujours en peer to peer, il ne peux traiter qu'un nombre de participants limités. Il est toujours possible de faire des systèmes plus complexes avec des super-noeuds mais cela augmente la complexité coté client. Enfin, se pose la question de décider quel point terminal sera le mélangeur.



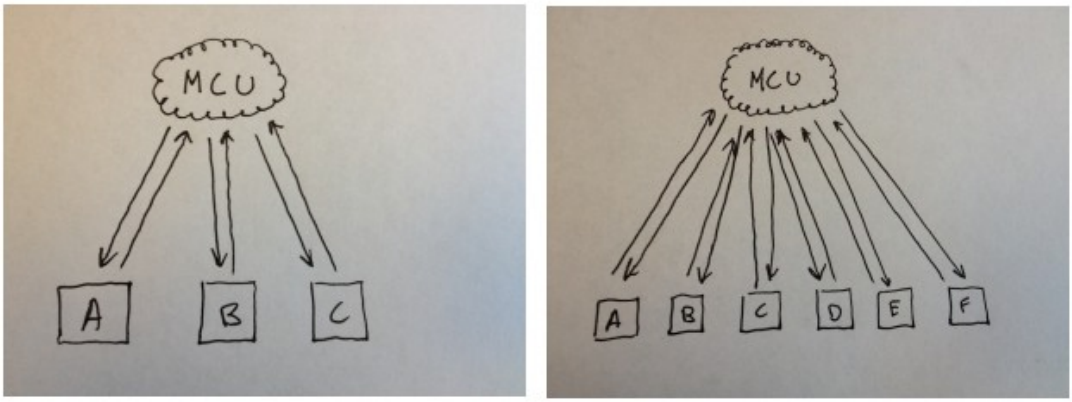
Topologies en étoile et multi-étoiles:
le ou les points terminaux font office de mélangeurs.

Les topologies multi-étoiles scalent mieux, mais se pose toujours le problème de déterminer quel points terminaux serviront de mélangeur. Apparaissent aussi d'autres facteurs de complexité: Par exemple comment faire pour que le terminal B puisse couper le flux provenant du client F.

La meilleure option pour gérer un grand nombre de paramètres est de passer par un MCU[13]. Il s'agit d'un serveur qui fonctionne comme un pont pour distribuer les médias à un grand nombre de clients. Un MCU peut faire face à différentes résolutions, des codecs et des taux de trame au sein d'une conférence vidéo, gérer le transcodage, faire des re-directions de flux sélective, mixer ou enregistrer l'audio et et la vidéo. En bref, ce sont de véritables "serveurs de media" .

Plusieurs solutions logicielles de MCU open source sont disponibles. Par exemple, Licode (http://lynckia.com/) propose un MCU open source pour WebRTC, ou encore OpenTok avec Mantis (http://www.tokbox.com/blog/mantis-next-generation-cloud-technology-for-webrtc/) .

[13] Un **multipoint control unit** (**MCU**) est un logiciel informatique ou une machine servant à établir simultanément plusieurs communications de visioconférence ou de VoIP.



MCU: Multipoint Control Unit (alias "Serveur de médias")

Les avantages d'un MCU: Moins de calculs, pas d'opérations de mixage (mélange de flux), réduction très importante des besoins en bande passante.

A noter que dans le cadre du projet AZKAR, les besoins seront relativement limités: 1-1 ou au pire en 1-2 (visite musée) ou en N x N (visioconférence à plusieurs via WebRTC dans le scénario santé) mais probablement limitée à N = 3 ou 4. Comme nous l'avons déjà évoqué dans la partie 2.3.2, le problème se posera de manière plus épineuse dans le cas de multiples déploiements de robots...

2.4 Exemples d'applications, projets en cours

Il est a noter que tous les jours de nouveaux projets apparaissant, ces listes sont loin d'être exhaustives. Quelques solutions et "produits" clé en mains commerciaux ou open-source, basés sur WebRTC:

- jitsi.org (Projet opensource de webmeeting, MIT, renater)
- talky.io (Projet commercial de video conferencing)
- tawk.com (Projet commercial de video conferencing)
- free.gotomeeting.com (Projet commercial de webmeeting)
- codassium.com (Projet commercial: plateforme d'interview et de live coding)
- appear.in (Projet commercial de video conferencing)
- vmux.co (Projet commercial de video conferencing)

Quelques projets de contôle à distance de robots par internet :

- BeeBot, projet collaboratif de plusieurs universités indonésiennes: Robot de téléprésence utilisant WerRTC et Google App Engine. https://www.youtube.com/watch?v=i6cF131f5a0

- Un projet de l'institut de recherche allemand Fraunhofer FOKUS utilisant OpenMTC (middleware M2M open source - M2M = MachineToMachine) pour les commandes du robot et WebRTC pour l'appel vidéo entre le robot et le dispositif de commande. https://www.youtube.com/watch?v=5UVGMQ2BZQA

- Petit projet amateur (William Cooley): Un robot de télé-présence utilisant les data channels de WebRTC pour envoyer des commandes au moteur du robot avec un faible latence.https://www.youtube.com/results?search_query=WebRTC+robot

- Encore un autre projet amateur basé sur WebRTC et socket.io https://www.youtube.com/watch?v=TrFTjqgk0hc

- Une démo de course de robots lors d'une conférence WebRTC au japon https://www.youtube.com/watch?v=oO-WjCKX9LY&feature=youtu.be

Autres projets tournant autour de WebRTC :

- Expérience d'interconnexion multi-plateformes (Google Glass, Androïd, IOS) avec WebRTC. https://www.youtube.com/watch?v=1w95PJLr8V4

2.5 Frameworks, librairies et projets de développement

Au 9/02/2015, Github[14] comporte pas moins de 3627 projets utilisant WebRTC.

La partie "mise en relation des pairs", que nous avons défini dans la section 2.3.1 comme étant "la phase de signalisation"', fait partie du standard WebRTC, mais aucune indication concernant son implémentation n'est présente dans la spécification. C'est aux développeurs de l'implémenter. De même la partie "TURN" qui indique comment utiliser un serveur de relais dans le cas où la communication pair à pair ne peut être établie, ne fait l'objet d'aucune indications relatives à son implémentation.

Il est assez simple d'implémenter la partie "signalisation". On trouve de nombreux exemples et tutoriels sur internet, montrant comment implémenter la signalisation via des WebSockets, via Ajax, via la "channel API" de Google App Engine (la solution de cloud de Google), via les services web d'Amazon, via FireBase, via Meteor, etc.

On trouve également des librairies de plus haut niveau simplifiant le développement d'applications WebRTC au sens large, dans le monde open source mais aussi commerciales. Le plus souvent, dans le cas des produits commerciaux, la librairie est gratuite mais l'utilisation des services de signaling ou de TURN est payant (ce dernier point nécessitant de grosses infrastructures pour scaler, lorsque le nombre de connexions vidéo devient important, on retrouve le plus souvent les opérateurs télécom ici).

2.5.1 Librairies open source (y compris commerciales):

- **PeerJS** (https://github.com/peers/peerjs) propose aux développeurs un outil capable de créer des connexions pair à pair de manière simple. Présenté sous la forme d'une bibliothèque JavaScript, il fonctionne tel un wrapper pour WebRTC et autorise ainsi la création de connexions à l'aide de seulement quelques lignes de code. PeerJS négocie des connexions avec WebRTC et autorise les connexions par identifiant de pairs, mais nécessite cependant un serveur de signaling avec Node.js.

- **EasyRTC** (http://easyrtc.com/) se concentre principalement sur les conférences vidéo 1-1 mais aussi N x N. En raison de sa facilité d'installation et d'utilisation, il à déjà été déployé sur plus de 5000 serveurs de test et aussi en production. Cette bibliothèque propose d'utiliser des serveurs de signaling (STUN) et TURN publics, configurables. Elle inclut également le transfert de fichiers en pair à pair et l'établissement des communications de manière sécurisée via un système de Tokens.

[14] https://github.com/search?p=100&q=webrtc+in%3Aname%2Cdescription%2Creadme&type=Repositories

- **webRTC.io** (https://github.com/webRTC-io/webRTC.io) est une couche d'abstraction pour WebRTC. Le but est de simplifier l'implémentation de WebRTC dans HTML5 d'une manière similaire à socket.io pour l'intégration des Web Sockets. Ce projet en est encore à un stade précoce.

- **rtc.io** (https://rtc.io/) est une autre librairie populaire, assez simple à utiliser. Utilise des serveurs STUN et TURN publics. Supporte les communications N x N mais pas le transfert de fichier pair à pair.

- **SimpleWebRTC** (https://simplewebrtc.com/) est certainement la plus ancienne des librairies WebRTC semble avoir perdu sa popularité aujourd'hui.

- **WebRTCmulticonnection** (http://www.rtcmulticonnection.org/), un des WebRTC experiments. Propose aussi une surcouche à WebRTC pour les communications 1-1 mais aussi N x N avec gestion de salles virtuelles. Utilise un serveur FireBase pour le signaling.

Solutions commerciales :

On trouve également de nombreuses librairies de haut niveau dans le monde commercial, simplifiant l'écriture d'applications utilisant la technologie WebRTC. Le plus souvent, elles proposent également des solutions SAAS pour les serveurs STUN (signaling), TURN (serveur relais), le stockage des sessions enregistrées, des systèmes de stockages pour fichiers partagés (à la Google Drive / DropBox, etc.), des systèmes d'identification paramétrables, etc.

Certaines librairies JavaScript, comme OpenTok (propriété de Telefonica, un opérateur télécom espagnol), proposent également des SDK pour développer des applications natives, reprenant la même démarche de développement que les APIs JavaScript, et partageant les mêmes services.

Parmi celles-ci, outre EasyRTC déjà cité, on trouve[15] AddLive (racheté par Snapchat), Apidaze, Apizee, CafeX, Forge (d'Acision), OpenClove, Plivo, Requestec, SightCall (anciennement Weemo), Sinch, Temasys, TokBox, Tropo, Twilio, VoxImplant, etc...

[15] Source: https://bloggeek.me/choosing-webrtc-api-platform/

Le cas particulier d'iOS

Jusqu'à présent Apple n'autorise pas les navigateurs tiers à utiliser leur propre moteur de rendu HTML5/JavaScript. Ainsi, qu'on soit sur Google Chrome ou sur Safari sur un iPhone ou un Ipad, on aura les mêmes limitations: si Safari ne supporte pas les APIs WebRTC, alors Chrome ne le supportera pas non plus !

Safari ne supportant pas, pour des raisons économiques et stratégiques WebRTC (Apple possède FaceTime), la seule solution pour faire du WebRTC sur IOS consiste à développer des applications natives. On trouve plusieurs SDKs permettant d'y arriver, tels que Bistri (https://bistri.com/) ou OpenTok (https://tokbox.com/opentok/).

De même, certains plugins pour navigateurs, tels ceux de Temasys (https://www.temasys.com.sg/) permettent d'activer WebRTC sur Safari, mais aussi sur des navigateurs de la famille IE.

Toutefois, l'absence de prise en charge par Safari ne semble pas constituer un frein important a l'essor de WebRTC: En effet, plusieurs représentants d'Apple sont maintenant membres du Working Group au W3C[16].

[16] Source: http://www.macg.co/logiciels/2014/12/interview-webrtc-le-standard-de-communication-prometteur-encore-meconnu-86179

2.6 Tests préliminaires

Nous avons réalisé quelques tests préliminaires de certaines API et solutions commerciales :

- Adobe Connect comme solution de référence et outil de coordination en parallèle.
- 2 librairies dévelopées en interne (Mypeerconnection & Codelab7).
- 3 API commerciales (Opentok, bistri et apizee)
- 1 API open source (simpleWebRTC)
- 1 Solution intégrée au navigateur (hello de firefox)
- 5 Solutions commerciales:
    - appear.in
    - tawk.com
    - gotomeeting
    - codassium (solution de livecoding)
- 1 solution OpenSource (Jitsi-meet)

Cette séance de tests préliminaires s'est révélée riche d'enseignements: Tout d'abord WebRTC à globalement démontré une meilleure réactivité (latence plus faible) qu'Adobe Connect, ainsi qu'une bonne, voire meilleure qualité de rendu (capacités de HD). D'autre part, les gains observés en matière de bande passante semblent en grande partie obtenus en diminuant les paramètres de définition (les "constraints") des objets Stream de l'API WebRTC. Il faudra donc ajuster cette définition de manière optimale et individuelle dans les maquettes en fonction des besoins des scénarii.

De plus, plusieurs solutions se sont nettement détachées des autres, tant pour leurs performances que pour leur ergonomie: appear.in et jitsi-meet. Le principal intérêt de cette dernière est qu'elle est Open Source. Ce sont de bons candidat à étudier et a intégrer dans nos benchmarks...

D'autre part, les enseignements méthodologiques (l'utilisation d'une solution de référence et de coordination, mieux cerner les fonctionnalités à tester et les contextes d'essais, etc...) seront très utiles dans les phases ultérieures du projet.

Enfin, étant donné que pour des raisons de sécurité il est impossible d'autoriser à distance l'ouverture des micros et caméras dans les navigateurs classiques, il faudra veiller a utiliser une version stabilisée du navigateur installé sur le robot (Chromium ou Firefox) et, pour annuler cette contrainte:

- Soit lancer le navigateur avec certaines options en ligne de commande.
- Soit utiliser une version recompilée du navigateur sans cette contrainte.

3 Benchmarking / Mesures de performance : étude préliminaire

Pour les mesures de performances, il est nécessaire de prendre en compte :

1. **La qualité intrinsèque du réseau** (latence, bande passante) entre les deux points de mesure (adresses IP des deux pairs qui vont communiquer), cette qualité peut varier dans le temps (différentes périodes dans la journée, de la semaine) mais également en fonction de la nature du réseau (3G/4G, abonnements grands publics ou professionnels avec garantie de service, wifi public/privé, plus ou moins sécurisé, Ethernet, etc.).

2. **La qualité des implémentations de la couche WebRTC de bas niveau dans les différents navigateurs** sur une même machine et un même OS.

3. **La qualité de l'implémentation des applications WebRTC** que nous allons mesurer. Pour une même tâche (par exemple : visioconférence et échanges de données en aller/retour), il est possible d'aller de l'implémentation de bas niveau (à l'aide des fonctions de base des APIs WebRTC), à l'implémentation de très haut niveau (où on écrira l'application en quelques lignes de code pour la partie WebRTC, à l'aide d'une bibliothèque de haut niveau, il en existe plusieurs commerciales ou non). Les performances pourront être différentes.

4. **La qualité du hardware des machines en communication**. On parle ici de la résolution des caméras, du CPU, des GPUs, etc. Le flux vidéo étant encodé / décodé lors de l'envoi / réception, la puissance des processeurs et puces d'encodage/décodage va avoir un impact, de même que la quantité d'information à communiquer qui dépend de la résolution.

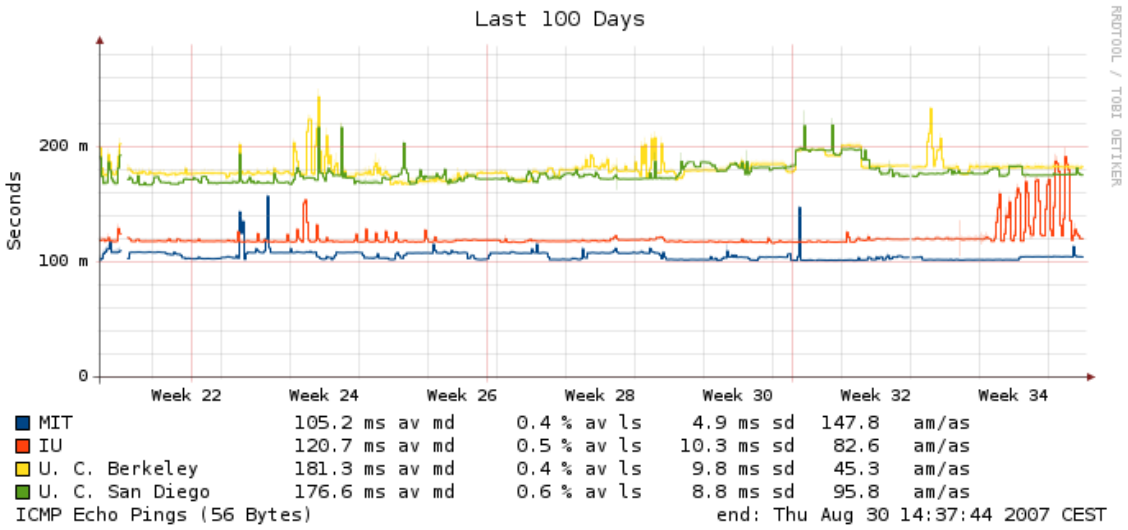
Pour la partie "mesure intrinsèque de la qualité du réseau", il existe plusieurs outils bien connus des ingénieurs systèmes, tels que :

- Testeur certificateur de réseau filaire pour mesurer la qualité du réseau sur lequel on est directement connecté, c'est le genre de matériel utilisé à l'Université pour voir si tout va bien sur le réseau Ethernet et vérifier qu'il n'y a pas de paquets perdus inutilement.

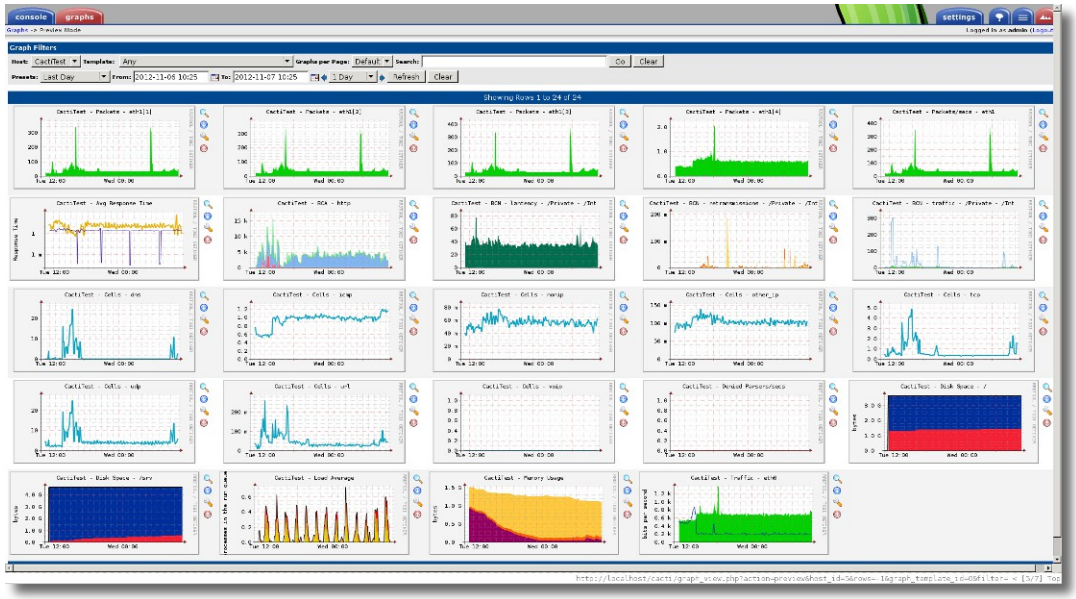
- Pour la latence entre deux IPs, l'outil de base est la commande "ping". Un outil graphique tel que smokeping (http://oss.oetiker.ch/smokeping/) peut également faciliter la mesure et l'établissement de rapports d'analyse.

- Pour le débit maximal entre deux points, l'outil iperf (https://iperf.fr/) est une référence (ne pas se fier aux tests de bande passante grands publics qui appartiennent tous à des opérateurs télécom et qui sont souvent biaisés).

- Pour faire des mesures de bande passante ou de latence moyenne sur une période donnée, on utilisera des outils comme mrtg (http://oss.oetiker.ch/mrtg/) ou cacti (http://www.cacti.net/)



Screenshot de mesures de latence par l'outil SmokePing



Screenshot de quelques visualisations de mesures de la qualité d'une connexion point à point avec l'outil Cacti

Pour la partie "mesure de qualité des implémentations de bas niveau WebRTC", on ne peut que faire des mesures comparatives avec une même application et une même configuration matérielle, lors de communication audio/video/data avec un pair de référence. La seule chose qui change d'une expérimentation à l'autre étant le navigateur. Des outils ont été développés, que nous avons identifiés :

- **WebRTC-benchmark** (https://github.com/FruitieX/webrtc-benchmark), permet de mesurer une implémentation entre deux navigateurs identiques. L'outil propose une application clé en main sur NodeJS, qu'il suffit de lancer localement. Des mesures quantitatives sont alors effectuées indépendamment du réseau (puisqu'on lance les deux pairs sur la même machine).

- **WebRTCBench** (http://sourceforge.net/projects/webrtcbench/) : une suite de benchmarks proposant une comparaison quantitative des implémentations dans les différents browsers. Inclut une base de données pour stocker les résultats des mesures et les consulter par la suite de manière ergonomique (avec tableaux, charts, etc). Ce benchmark a été développé par le "Parallel Architectures and Systems Lab" de l'Université de Irvine, en Californie.

Pour les points 3 et 4 (qualité de la couche d'implémentation des applications et qualité du hardware de la machine cliente), nous allons procéder comme suit :

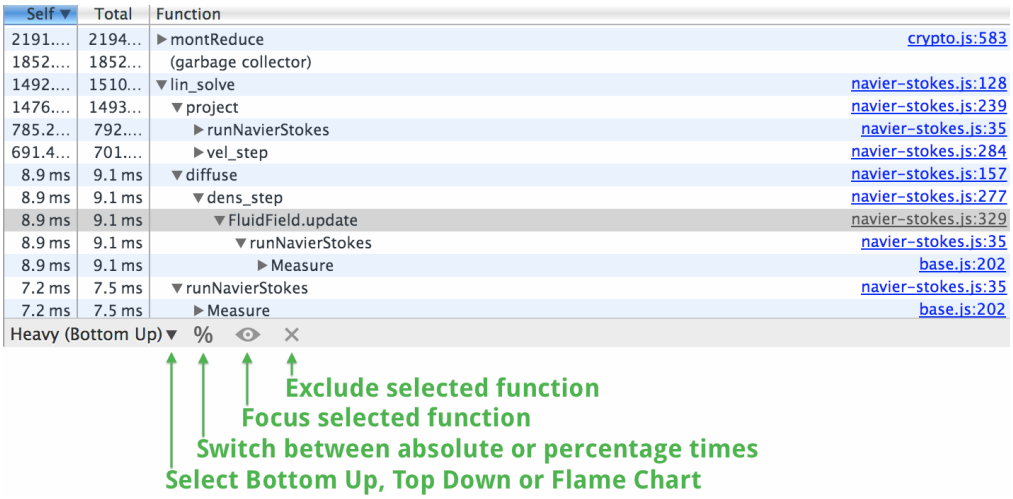
- Développer trois ou quatre versions d'une même application (streaming audio/video + data en 1-1 et en N x N), chaque version utilisant une approche différente : code 100% bas niveau, ou bien basé sur deux ou trois librairies de haut niveau parmi les plus populaires, commerciales ou non (exemple : EasyRTC, OpenTok, rtc.io)

- Tester cette application (ses trois ou quatre variantes) avec un pair de référence, sur différents types de réseaux (ceux dont nous aurons mesuré au préalable la performance intrinsèque).

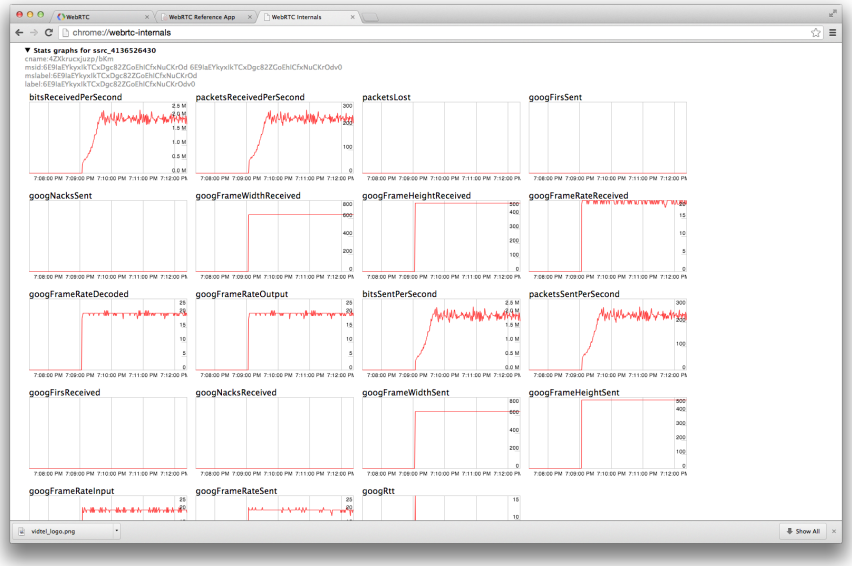
Nous procéderons à des mesures quantitatives (nombre d'images par seconde, taux de compression, latence, etc) et qualitatives (qualité du son et de la vidéo, réactivité ressentie, temps d'attente pour établir une connexion, bugs ?).

3.1 Mesures quantitatives

Pour les mesures quantitatives, des outils permettant de mesurer le temps de certaines tâches typiques seront mis en oeuvre, soit manuellement (installation de timers JavaScript dans le code) soit via des outils comme les profilers JavaScript inclus dans les devtools des différents navigateurs (pour mesurer que prennent certaines fonctions à s'exécuter), soit via des outils intégrés aux navigateurs, spécialisés pour la mesure des applications WebRTC (comme l'outil chrome://webrtc-internals).



Outil de profilage des applications JavaScript (devtools de Google Chrome)



Outil intégré à Google Chome pour mesurer les performances des applications WebRTC.

Cas de l'usage d'un serveur TURN : dans certains cas, la communication pair à pair ne peut avoir lieu et il est nécessaire de passer par un serveur de relais (serveur TURN). Dans ce cas les échanges sont forcément ralentis. Le temps perdu est pris par la nécessité de relayer les informations via un serveur distant, par le fait que ce serveur peut être plus ou moins chargé (il peut saturer s'il doit relayer trop de communications simultanément), par la qualité hardware de ce serveur, par la qualité du réseau entre les deux pairs et ce serveur, etc. Il sera important, dans le cadre du projet AZKAR d'arriver à identifier les cas où la communication passe par un serveur TURN et comparer les différences par rapport à la même expérience en pair à pair.

La latence "acceptable", à quoi est-ce que cela correspond ?

Tout dépend de ce que l'on désire faire à travers la connexion.

Dans le cas du contrôle d'un robot à distance à faible vitesse, Robosoft et la communauté des roboticiens parlent d'un temps de ping (temps pour envoyer un message et qu'il revienne) qui doit être inférieur à 100ms pour avoir un boucle de contrôle viable. Et encore, cette affirmation est très imprécise car il manque de nombreux détails tels que : vitesse de déplacement, moyens dont on dispose pour compenser la latence, pour "sentir" le robot dans son environnement distant, etc.

Dans le cadre du jeu vidéo d'action multi-joueurs, des mesures ont été effectuées dans le cadre de plusieurs cours d'informatique pendant lesquels des élèves de Master 1 et 2 de l'Université de Nice ont développé des jeux web en HTML5/JavaScript multi-participants. Pour implémenter la couche réseau les élèves se sont reposés sur la technologie des Web Sockets, qui implique forcément un serveur relais entre les clients connectés.

Sur un réseau local, et sur un ordinateur moderne, on obtient des temps aller-retour pour les échanges de messages de l'ordre de **200 microsecondes (0.2 ms)** à travers des Web Sockets (ce temps inclut l'envoi de données de moins de 1k à partir du code JavaScript tournant dans un navigateur, la réception via des Web Sockets par un serveur, et leur broadcast à plusieurs clients/navigateurs connectés), ce qui est similaire aux pings ICMP entre machines, hors application. **En d'autres termes, le coût de l'émission et du broadcast de ces messages est négligeable, presque imperceptible**.

Sur un réseau plus massif de type MAN (somme de réseaux locaux) mais dans une même infrastructure, par exemple sur l'intranet filaire de l'Université de Nice, dans des conditions d'usage du réseau normal, ce temps est autour de 10 ms, ce qui permet de jouer à des jeux d'action sans algorithmes de compensation de latence, de manière très confortable,
aucun temps de retard n'étant perceptible par des humains.

Sur un WAN (ADSL résidentiel au serveur dans un même pays) et avec une bonne connexion (4-10 Mbits/s) tests effectués chez des particuliers avec des connexion via opérateurs ADSL français classiques) ce temps passe autour de 30ms.

Avec la 3.5/4G le temps monte à 120-200ms ce qui n'est pas insurmontable mais nécessite l'implémentation d'algorithmes de compensation de latence pour ne pas ressentir "d'effet de lag".

Ce qu'il faut retenir: **la communication de faible quantités de données par Web Sockets ne rajoute pratiquement aucun temps de latence par rapport à celle que l'on a de toutes façons, sur la base du réseau, sans faire tourner l'application.**

Hypothèse de travail concernant le passage en WebRTC : nous supposons pour ce projet que la latence sera inférieure à celle mesurée avec des Web Sockets, pour l'échange de données seule, puisqu'on s'affranchi d'un serveur de relais (sauf dans le cas "dégradé" où on doit passer par un serveur TURN). Il sera intéressant de mesurer quel impact l'ajout de la vidéo et de l'audio temps réel aura sur les temps mesuré. Il sera intéressant de comparer les mesures effectuées avec des Web Sockets + une vidéo conférence en WebRTC par dessus, en mode pair à pair, avec la même solution 100% réalisée en WebRTC.

Algorithmes de compensation de latence pour le contrôle à distance

Il s'agit d'un problème très semblable à celui rencontré dans le domaine des jeux vidéo. On trouve de nombreux algorithmes, avec explications et implémentations, dans le domaine public. Par ailleurs, la littérature scientifique concernant la compensation de latence pour le contrôle de robots est assez fournie.

Le projet AZKAR vise, en fonction des résultats de la phase de benchmark, à implémenter un algorithme de ce type dans la couche de contrôle à distance du robot.

Ressources :

- http://blog.lightstreamer.com/2013/10/optimizing-multiplayer-3d-game.html
- Techniques to use in a real-time web stack : http://www.blogger.com/blogger.g?blogID=2020370271027153869#Techniques
- http://buildnewgames.com/optimizing-websockets-bandwidth/
- https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking
- ...

3-2 Mesures qualitatives (Spécification de la méthode d'évaluation qualitative)

La tâche « Mesures qualitatives » a pour objectif général d'évaluer, du point de vue de l'utilisateur, la qualité de la transmission des informations entre dispositifs AZKAR robotisés (robot Kompaï, robuCITY, robuLab10) d'une part et les dispositifs de contrôle à distance des dispositifs robotisés d'autre part.

Une évaluation du point de vue de l'utilisateur requiert en principe de prendre en considération les caractéristiques de cet utilisateur, ses buts, ses tâches, l'environnement dans lequel il réalise ces tâches, les dispositifs qui l'aident à réaliser ses tâches, etc. Bref une évaluation du point de vue de l'utilisateur devrait être une évaluation en situation (en situation « écologique ») ; elle devrait être une évaluation contextuelle ou contextualisée. L'évaluation devrait donc être guidée le plus possible par les scénarios d'usage envisagés dans le projet AZKAR : scénario Transports, scénario Culture et scénario Santé. Ces scénarios présentant certaines spécificités, ces dernières devraient se refléter dans la définition de l'objectif de la tâche « Mesures qualitatives » et, de façon plus globale, sur l'ensemble de la méthodologie d'évaluation qualitative.

Le but de cette section est d'amorcer ce travail de spécification de l'objectif et de la méthode d'évaluation (des critères qualitatifs en particulier). On propose ici un cadre pour réaliser cette spécification. Ce cadre s'inspire de la méthode expérimentale.

3-2.1 Rappel des scénarios AZKAR

- ***Scénario Transports*** (ramassage de linge à distance)
    - Description issue de l'annexe technique du projet AZKAR
        - Un opérateur distant (installé dans un local de supervision) contrôle à distance un véhicule robotisé permettant le ramassage de linge. L'opérateur intervient en particulier dans les situations de blocage du véhicule.

- ***Scénario Culture*** (visite d'un musée à distance)
    - Description issue de l'annexe technique du projet AZKAR
        - Visite avec médiation
            - Visite d'une exposition en diffusion
            - Conférence itinérante
            - Partage d'une visite de groupe
            - Visite avec un public restreint permettant le dialogue
        - Visite sans médiation
            - Prise en main du robot par un internaute

    - Description issue d'un point téléphonique AZKAR
        - Cas 1 : l'internaute se promène dans le musée via le robot
        - Cas 2 (conférence itinérante, dans une maison de retraite) : le conférencier, dans la maison de retraite, commande un robot sur un site distant
        - Cas 3 : le conférencier, chez lui, face aux internautes
        - Cas 4 (dit de l'Avatar) : le robot se joint à une visite de groupe sur site, et le robot est connecté à des personnes extérieures

- ***Scénario Santé*** (consultation à distance)

    - Description issue d'un point téléphonique AZKAR
        - Démo santé : scénario de téléconsultation intégrant la télémétrie. L'aidant professionnel se pose une question sur le patient et manipule le robot ; il récupère des informations à distance.

Ces scénarios sont à l'heure actuelle à l'état d'ébauches. Ils seront précisés par les partenaires chargés des évaluations de terrain.

La question est de savoir dans quelle mesure les scénarios vont guider l'évaluation. Quel degré de contextualisation sera appliqué ? Les extrêmes sont, d'un côté, une évaluation fortement contextualisée (les vrais utilisateurs réalisant une tâche réelle dans des conditions réelles) et, de l'autre côté, une évaluation hors contexte (on ne prend pas des utilisateurs cibles, on ne leur fait pas réaliser des tâches réelles, etc.).

Une évaluation fortement contextualisée étant coûteuse, pour une évaluation initiale moins coûteuse, on envisagera un quatrième scénario où le contexte sera simplifié. Ce scénario reprendra des éléments des trois scénarios précédents. Il s'agit du :

- ***Scénario Développement***, dans lequel les développeurs joueront le rôle d'utilisateurs standards et évalueront la qualité de la transmission des informations entre les dispositifs AZKAR robotisés et les dispositifs de contrôle à distance des dispositifs robotisés.

La description qui suit du cadre méthodologique est faite en fonction de ces quatre scénarios.

3-2.2 Cadre méthodologique

Objectif de l'évaluation. Préciser l'objectif de l'évaluation en fonction de chaque scénario :

- Pour le scénario Transports :
  - Focalisation sur le contrôle à distance du véhicule robotisé.

- Pour le scénario Culture :
  - *A préciser*.

- Pour le scénario Santé :
  - Focalisation sur les « fonctions de communication et de télé-présence (faire disparaître les temps de latence et les divers retards qui perturbent la qualité de l'assistance à distance) ». L'objectif est de comparer la « téléopération robotisée via Kompaï vs autres technologies (visioconférence, visiophone, téléphone, etc.)».

- Pour le scénario Développement :
  - Effectuer une évaluation préliminaire de la qualité perçue des transmissions et de la télécommande des dispositifs robotisés.

Participants à l'évaluation. Préciser les participants à l'évaluation :

- Pour le scénario Transports :
  - Opérateur à distance

- Pour le scénario Culture :
  - Visiteurs
  - Conférencier

- Pour le scénario Santé :
  - Personnes âgées (avec handicap physique léger uniquement, pas de handicap mental ; vivant seules à domicile bénéficiant de l'assistance d'aides humaines). Dans l'annexe technique du projet AZKAR, il est indiqué que seront sélectionnés « 5 candidats volontaires, très faiblement dépendants et vivant à proximité de CENRob, à partir d'un focus group organisé par le CENRob »
  - Aidants : aidants professionnels | aidants familiaux

- Pour le scénario Développement :
  - Développeurs AZKAR (I3S, Robosoft, AW)

Hypothèses et variables indépendantes et contrôlées. Préciser les hypothèses à tester :

- Pour le scénario Transports :
  - *A préciser*.
- Pour le scénario Culture :
  - *A préciser*.
- Pour le scénario Santé :
  - *A préciser*.
- Pour le scénario Développement (exemple) :
  - La solution WebRTC est de meilleure qualité que les autres solutions existantes (ou tout au moins de qualité comparable).

Pour aider à préciser les hypothèses, identifier les variables indépendantes (celles qui auront une influence sur les mesures), mais aussi les variables à contrôler. Par exemple :

- Type d'utilisateur /acteur
- Type de dispositif : tablette | téléphone (smartphone) | ordinateur
- Type de liaisons : MAN | WAN | 3G | 4G | Wifi
- Type de navigateur : Firefox | Chrome | Safari | …
- …

Matériel. Préciser le matériel qui sera utilisé lors de l'évaluation :

- Pour le scénario Transports : **robuCITY** (véhicule pour le Transports automatique de personnes) (ou ordinateur simulant le robot) ; dispositif de manipulation du robot à distance.

- Pour le scénario Culture : **robuLab10** (ou ordinateur simulant le robot) ; tablette ; ordinateur ; téléphone (smartphone).

- Pour le scénario Santé : **robot Kompaï** (ou ordinateur simulant le robot), réseau Wifi.

- Pour le scénario Développement : robuCITY, robuLab10, Kompaï (ou ordinateur simulant le robot), dispositif de manipulation du robot à distance.

Lieu des tests. Préciser dans quel (s) endroit(s) se déroulement les tests :

- Pour le scénario Transports :
  - Les évaluations seront faites sur un site réel.
- Pour le scénario Culture :
  - Universcience (La Villette) : retours d'usage à travers le living lab du Carrefour numérique2.
  - Musée de la Grande Guerre (Meaux) : espaces muséaux.
- Pour le scénario Santé :
  - Locaux du CENRob pour l'initiation.
  - Domicile des P.A. ensuite.
- Pour le scénario Développement :
  - Labos des Développeurs.

Tâches. Préciser les tâches qui seront réalisées par les utilisateurs lors de l'évaluation :

- Pour le scénario Transports :
  - Tâche de l'opérateur distant (installé dans le local de supervision) : « prendre la main sur le véhicule pour l'arrêter en cas de problème » ; « dégager à distance le véhicule si nécessaire pour débloquer la situation (le véhicule a heurté accidentellement un obstacle et bloque la voie, d'où la nécessité de le dégager rapidement pour libérer la voie de circulation) ».
- Pour le scénario Culture :
  - Tâche du conférencier : *A définir*.
  - Tâche du visiteur distant : piloter le robot.
- Pour le scénario Santé :
  - « Mise en place d'un planning journalier de surveillance alternée (aides humaines sur site, aides humaines via le robot Kompaï) ».
- Pour le scénario Développement :
  - Tâches simplifiées. *A préciser.*

Durée des tests. Préciser la durée des tests :

- Pour le scénario Transports :
  - *A préciser*.
- Pour le scénario Culture :
  - Universcience. Quatre sessions de retours d'usage prévues, sous forme de focus group, durant les 9 mois de conduite des évaluations.
  - Musée de la Grande Guerre : *à préciser*.
- Pour le scénario Santé :
  - Durée de l'étude observationnelle : 30 jours.
- Pour le scénario Développement :
  - *A préciser*.

Mesures/Critères (variables dépendantes)

Préciser les mesures/critères (variables dépendantes) qui seront utilisés pour évaluer la solution AZKAR. (Pour les scénarios Transports, Culture et Santé, on rappelle les critères mentionnés dans divers documents AZKAR.)

- Pour le scénario Transports :
  - Critères mentionnés dans l'annexe technique du projet AZKAR
    - Qualité
    - Latence (pouvoir piloter le véhicule sans aucune latence tout en récupérant des images en live)
    - Performance des algorithmes

- Pour le scénario Culture :
  - Critères mentionnés dans l'annexe technique du projet AZKAR
    - Qualité du flux vidéo (mise en valeur du site ou de l'espace muséal visité)
    - Flux vidéo complété par une image vidéo du médiateur et/ou une vidéo du public distant (dans le cas d'une diffusion restreinte)
    - Simplicité de la conduite du robot
    - Qualité de la restitution visuelle pour le public distant

- Pour le scénario Santé :
  - Critères mentionnés dans l'annexe technique du projet AZKAR
    - Bénéfices ressentis
    - Grille de qualité de vie
    - Niveau de satisfaction [équivalent à la Qualité d'intervention de l'aide à domicile ?]
    - Niveau d'anxiété
    - Niveau de participation sociale
    - Usages programmés vs usages non programmés
    - Acceptabilité (utilisateurs, aidants professionnels, aidants familiaux)
  - Critères mentionnés dans les transparents powerpoint réalisés par Approche après le startup-meeting en Décembre 2014.
    - Critère principal :
      - la « Plus-value télé interaction robotisée via Kompai vs autres technologies (visioconférence, visiophonie, téléphonie, etc.) »
    - Critère secondaire
      - Utilisabilité
      - Robustesse
      - Réactivité

- Pour le scénario Développement (exemples[1]) :
    - Qualité audio (son) – Qualité perçue par l'utilisateur/développeur
        - *Confort :* inconfortable | confortable | très confortable
        - *Clarté/netteté/résolution :* mauvaise | passable | bonne |très bonne
        - *Volume sonore :* insatisfaisant | satisfaisant | très satisfaisant
        - *Continuité :* intermittent/continu
        - *Intégrité :* partielle |complète
        - *Manipulation du son :* médiocre | passable | bonne | très bonne
        - *Autres critères* fournis par l'utilisateur/développeur
    - Qualité vidéo (image) - Qualité perçue par l'utilisateur/développeur
        - *Confort :* inconfortable | confortable | très confortable
        - *Clarté/netteté/résolution/ définition :* mauvaise | passable | bonne |très bonne (ou) très basse |basse | moyenne |haute | très haute
        - *Fluidité de l'image :* très saccadé | saccadé | fluide | très fluide
        - *Manipulation de l'image :* médiocre | passable | bonne | très bonne
        - *Autres critères* fournis par l'utilisateur/développeur
    - Qualité data - Qualité perçue par l'utilisateur/développeur
        - *Latence :* Forte | Moyenne | Faible
        - *Volume des données transmises :* insuffisant | suffisant
        - *Manipulation des données :* médiocre | passable | bonne | très bonne
        - *Autres critères* fournis par l'utilisateur/développeur
    - *Qualité globale de la téléopération du dispositif robotique*

3-2.2 Priorisation des évaluations et des développements

L'ordre de priorité de réalisation des évaluations et des développements serait à déterminé en fonction de la complexité de chaque composante fonctionnelle. Trois composantes fonctionnelles sont à évaluer dans le projet:

- Une composante Data: télécommande du robot (intégrité ; latence ; volume fichiers…).
- Une composante Audio/Vidéo (qualité, latence…).
- Une composante de Scalabilité/Complexité (Nombre, types d'acteurs : robot, humain ; rôles : rôles d'administrateur: télémanipulateur, conférencier, soignant ; rôles d'intervenants (ou agent): robot, médecin consultant, patient, visiteur de musée individuel, accompagnant de groupe de visiteurs (professeur, etc) ; rôles de spectateurs passifs: groupe de visiteurs de musée).

Il serait intéressant d'évaluer en premier la partie Transports, le nombre d'acteurs étant limité à deux. Ici la composante Vidéo n'implique pas nécessairement une bonne qualité vidéo, seule l'aspect de latence est important. Les acteurs de cette évaluation seraient : l'administrateur (le télémanipulateur) et l'agent (le robot - mode unicast).

En second lieu pourrait venir l'évaluation Santé, l'évaluation Transports précédente nous ayant permis de capitaliser le retour d'expérience pour la partie motrice et télémanipulation. A ce stade la complexification consisterait en, l'amélioration de l'aspect motricité (se situer et se déplacer dans un appartement, détecter et géolocaliser la personne âgée, et une montée en puissance légère de l'aspect scalabilité, mais avec une quantité d'acteurs relativement réduite, par exemple : l'administrateur (l'aidant principal) et les agents (le robot, le malade, le ou les médecins consultants - mode multicast). Comme il est peu probable de se retrouver avec une centaine d'aidants, à ce stade la scalabilité n'est à priori pas encore un facteur bloquant.

Enfin en dernier lieu viendrait l'évaluation Culture (visite de musée), puisqu'elle bénéficierait déjà de toutes les avancées précédentes en termes de mobilité/motricité et de complexité des rôles. C'est seulement à ce stade qu'il faudra veiller particulièrement à la qualité vidéo de l'image et du son (on imagine mal de voir la Joconde en basse définition). A ce stade la complexité des rôles devient plus importante mais surtout c'est ici que l'on peut sérieusement envisager des problèmes de scalabilité selon le nombre d'acteurs connectés à la visite: Administrateur (le guide conférencier) ; Agents (le robot, visiteur individuel, accompagnant de visiteurs - mode multicast) ; Spectateurs (groupe de visiteurs - en mode broadcast ?).

3-2.3 Conclusion sur la méthode d'évaluation qualitative

Dans cette section, nous avons amorcé un travail de spécification de l'objectif et de la méthode d'évaluation qualitative de la solution AZKAR. Nous avons proposé un cadre pour réaliser cette spécification. Ce cadre doit maintenant être discuté avec les partenaires AZKAR.
Cette discussion sur le cadre méthodologique de l'évaluation de la solution AZKAR doit être également l'occasion de bien cerner le périmètre du projet AZKAR afin d'aboutir à un périmètre réaliste.

---

[1] Certains de ces exemples sont inspirés des critères proposés pour tester les communications WebRTC dans Firefox (version 33) : *Qualité des appels* : honorable ; *Mise en relation* : rapide ; *Son* : correct (mais pas comparable avec Skype) ; *Image* : fluide ; ne saccade pas. On notera la présence d'un icône Firefox en haut et au centre de l'écran, pour ré-affichage de Firefox au premier plan. (Référence de l'étude : Vincent Hermann. Mozilla veut vous faire tester les communications WebRTC dans Firefox, 05/09/2014 ; http://www.nextinpact.com/news/89710-mozilla-veut-vous-faire-tester-communications-webrtc-dans-firefox.htm#page/2.)

4 Conclusion.

Comme nous l'avons vu, 2 technologies sont principalement en mesure de répondre a nos besoins RTC dans le cadre du projet AZKAR: Flash et WebRTC.

Flash de part son aspect propriétaire et la nécessité d'utiliser des plugins implique un investissement lourd, à la foi financier (achat de licences), technologique (développeurs spécialisés dans les technologies flash et plus particulièrement Adobe Connect), et pose un problème de pérennité et d'indépendance.

Par ailleurs, Flash ne propose pas de technologie de transfert Pair à Pair, or il est fort probable que les mesures de latence lors de la phase de mesure de performance, fassent ressortir une latence bien plus faible en connexion directe, par rapport à la solution Adobe Connect qui passe systématiquement par un serveur de relais.

Notons que si on est en pair à pair, sans passer par un serveur de relais (TURN dans le cas de WebRTC), alors le problème de la gestion de nombreuses connexions simultanées est simplifié, en faisant sauter le point central de relais, plus de goulot d'étranglement, plus besoin de solutions lourdes que seuls les opérateurs télécom peuvent fournir.

Enfin, même si les applications basées sur ces technologies propriétaires sont d'excellent produits, leur fonctionnement n'en reste pas moins relativement obscur.

D'autre part, on constate que les grands acteurs actuels du RTC, Google Hangout's, Skype (Microsoft), ont tendance à s'orienter clairement vers WebRTC. Microsoft vient d'ailleurs d'intégrer le groupe de travail WebRTC. On peu dès lors parier sans trop se tromper que WebRTC est promis a un bel avenir.